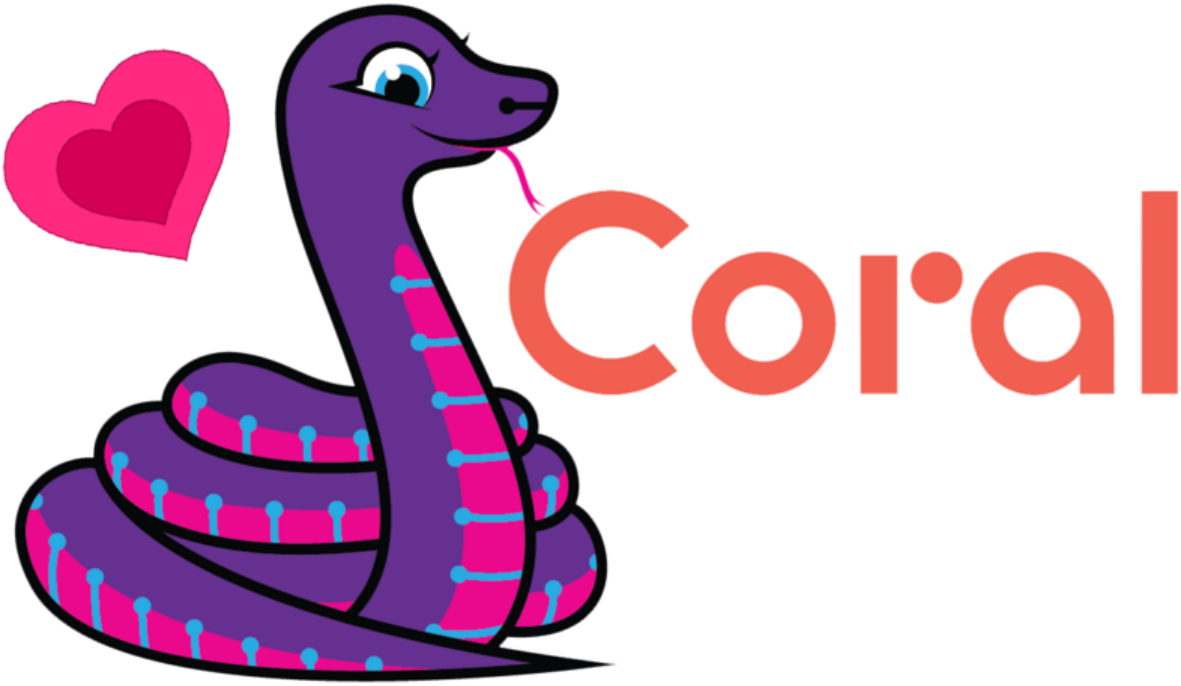
CircuitPython Libraries on Linux and Google Coral

Created by lady ada



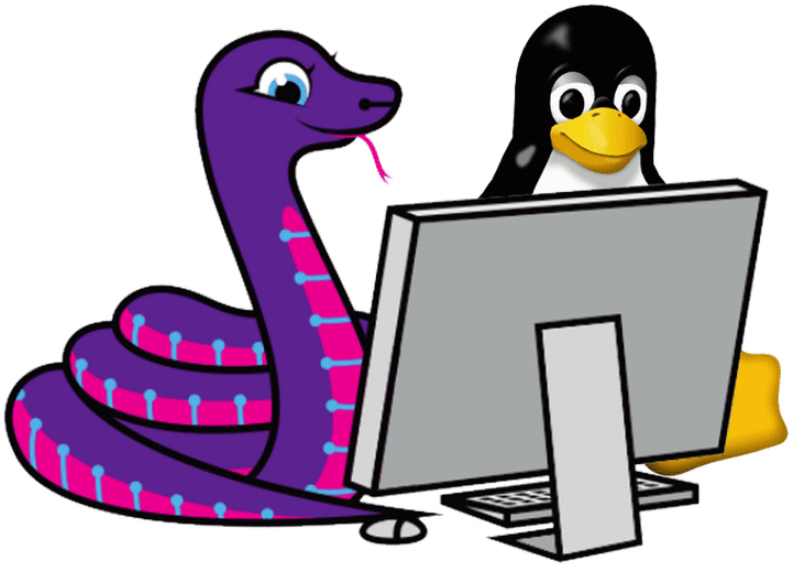
https://learn.adafruit.com/circuitpython-on-google-coral-linux-blinka

Last updated on 2025-10-01 04:12:57 PM EDT

Table of Contents

Overview

Please note! All the stuff in this guide works and we're improving and working on this code a bunch so be sure to check back for updates!



Here at Adafruit we're always looking for ways to make making easier - whether that's making breakout boards for hard-to-solder sensors or writing libraries to simplify motor control. Our new favorite way to program is **CircuitPython**.

Why CircuitPython?

CircuitPython is a variant of MicroPython, a very small version of Python that can fit on a microcontroller. Python is the fastest-growing programming language. It's taught in schools, used in coding bootcamps, popular with scientists and of course programmers at companies use it a lot!

CircuitPython adds the Circuit part to the Python part. Letting you program in Python and talk to Circuitry like sensors, motors, and LEDs!

CircuitPython on Microcontrollers

For a couple years now we've had CircuitPython for microcontrollers like our SAMD21 series with Feather/Trinket/CircuitPlayground/Metro M0, as well as the ESP8266 WiFi microcontroller, nRF52 bluetooth microcontroller and SAMD51 series.

All of these chips have something in common - they are microcontrollers with hardware peripherals like SPI, I2C, ADCs etc. We squeeze Python into 'em and can then make the project portable.

But...sometimes you want to do more than a microcontroller can do. Like HDMI video output, or camera capture, or serving up a website, or just something that takes more memory and computing than a microcontroller board can do...

Running CircuitPython Code without CircuitPython

There are two parts to the CircuitPython ecosystem:

- **CircuitPython firmware**, written in C and built to run on various microcontroller boards (not PCs). The firmware includes the CircuitPython interpreter, which reads and executes CircuitPython programs, and chip-specific code that controls the hardware peripherals on the microcontroller, including things like USB, I2C, SPI, GPIO pins, and all the rest of the hardware features the chip provides.
- **CircuitPython libraries,** written in Python to use the native (built into the firmware) modules provided by CircuitPython to control the microcontroller peripherals and interact with various breakout boards.

But suppose you'd like to use CircuitPython **libraries** on a board or computer that does not have a native CircuitPython **firmware** build. For example, on a PC running Windows or macOS. Can that be done? The answer is yes, via a separate piece of software called **Blinka**. Details about Blinka follow, however it is important to realize that the **CircuitPython firmware is never used**.

> CircuitPython firmware is NOT used when using Blinka.

Adafruit Blinka: a CircuitPython Compatibility Library

Enter **Adafruit Blinka**. Blinka is a software library that emulates the parts of CircuitPython that control hardware. Blinka provides non-CircuitPython implementations for `board`, `busio`, `digitalio`, and other native CircuitPython

modules. You can then write Python code that looks like CircuitPython and uses CircuitPython libraries, without having CircuitPython underneath.

There are multiple ways to use Blinka:

- Linux based Single Board Computers, for example a Raspberry Pi
- Desktop Computers + specialized USB adapters
- Boards running MicroPython

More details on these options follow.

Raspberry Pi and Other Single-Board Linux Computers

On a Raspberry Pi or other single-board Linux computer, you can use Blinka with the regular version of Python supplied with the Linux distribution. Blinka can control the hardware pins these boards provide.

Desktop Computers

On Windows, macOS, or Linux desktop or laptop ("host") computers, you can use special USB adapter boards that that provide hardware pins you can control. These boards include MCP221A (https://adafru.it/IfV) and FT232H (https://adafru.it/xia) breakout boards, and Raspberry Pi Pico boards running the **u2if** software (https://adafru.it/Sje). These boards connect via regular USB to your host computer, and let you do GPIO, I2C, SPI, and other hardware operations.

MicroPython

You can also use Blinka with MicroPython, on MicroPython-supported boards (https://adafru.it/SBi). Blinka will allow you to import and use CircuitPython libraries in your MicroPython program, so you don't have to rewrite libraries into native MicroPython code. Fun fact - this is actually the original use case for Blinka.

Installing Blinka

Installing Blinka on your particular platform is covered elsewhere in this guide. The process is different for each platform. Follow the guide section specific to your platform and make sure Blinka is properly installed before attempting to install any libraries.

> Be sure to install Blinka before proceeding.

Installing CircuitPython Libraries

Once Blinka is installed the next step is to install the CircuitPython libraries of interest. How this is down is different for each platform. Here are the details.

Linux Single-Board Computers

On Linux single-board computers, such as Raspberry Pi, you'll use the Python `pip3` program (sometimes named just `pip`) to install a library. The library will be downloaded from pypi.org (https://adafru.it/19ff) automatically by `pip3`.

How to install a particular library using `pip3` is covered in the guide page for that library. For example, here is the `pip3` installation information (https://adafru.it/OkF) for the library for the LIS3DH accelerometer.

The library name you give to `pip3` is usually of the form `adafruit-circuitpython-`*`libraryname`*. This is not the name you use with `import`. For example, the LIS3DH sensor library is known by several names:

- The GitHub library repository is Adafruit_CircuitPython_LIS3DH (https://adafru.it/uBs).
- When you import the library, you write `import adafruit_lis3dh`.
- The name you use with `pip3` is `adafruit-circuitpython-lis3dh`. This the name used on pypi.org (https://adafru.it/19ff).

Libraries often depend on other libraries. When you install a library with `pip3`, it will automatically install other needed libraries.

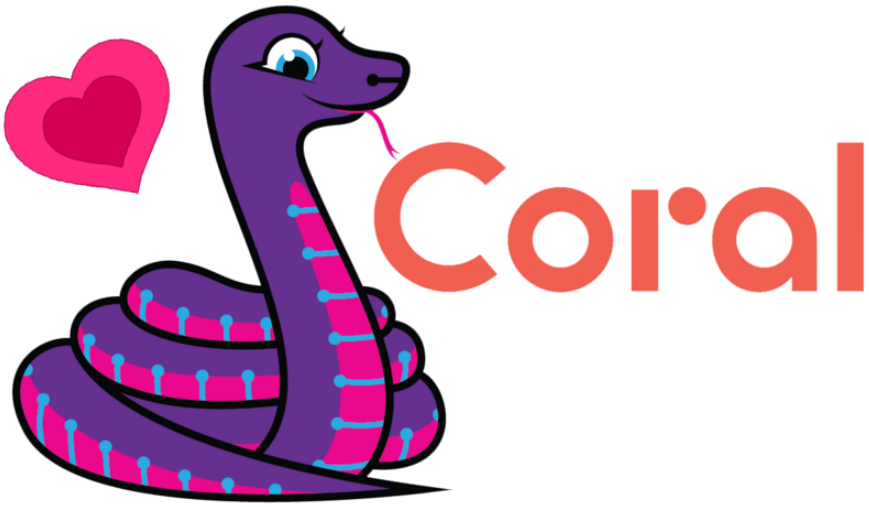
Desktop Computers using a USB Adapter

When you use a desktop computer with a USB adapter, like the MCP2221A, FT232H, or u2if firmware on an RP2040, you will also use pip3. However, **do not install the library with** `sudo pip3`, as mentioned in some guides. Instead, just install with `pip3`.

MicroPython

For MicroPython, you will not use `pip3`. Instead you can get the library from the CircuitPython bundles. See this guide page (https://adafru.it/ABU) for more information about the bundles, and also see the Libraries page on circuitPython.org (https://adafru.it/ENC).

CircuitPython & Coral



CircuitPython Libraries on Linux & Google Coral

The next obvious step is to bring CircuitPython ease of use **back** to 'desktop Python'. We've got tons of projects, libraries and example code for CircuitPython on microcontrollers, and thanks to the flexibility and power of Python its pretty easy to get it working with micro-computers like Google Coral or other 'Linux with GPIO pins available' single board computers.

We'll use a special library called **adafruit_blinka** (https://adafru.it/BJS) (named after Blinka, the CircuitPython mascot (https://adafru.it/BJT)) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides. For example, on Coral we use the python **libgpiod** bindings. For any I2C interfacing we'll use ioctl messages to the `/dev/i2c` device. For SPI we'll use the `spidev` python library, etc. These details don't matter so much because they all happen underneath the **adafruit_blinka** layer.

The upshot is that any code we have for CircuitPython will be instantly and easily runnable on Linux computers like Google Coral.

In particular, we'll be able to use all of our device drivers - the sensors, led controllers, motor drivers, HATs, bonnets, etc. And nearly all of these use I2C or SPI!

Wait, isn't there already something that does this - Periphery?

Periphery is a pure python hardware interface class (https://adafru.it/ENJ) for Coral, it works just fine for I2C, SPI and GPIO but doesn't work with our drivers as its a different API

By letting you use CircuitPython libraries on Raspberry Pi via **adafruit_blinka**, you can unlock all of the drivers and example code we wrote! And you can keep using **periphery** if you like. We save time and effort so we can focus on getting code that works in one place, and you get to reuse all the code we've written already.

What about other Linux SBCs?

Yep! Blinka can easily be updated to add other boards. We've started with the one we've got, so we could test it thoroughly. If you have other SBC board you'd like to adapt check out the adafruit_blinka code on github (https://adafru.it/BJX), pull requests are welcome as there's a ton of different Linux boards out there!

Initial Setup

Your Coral dev board comes blank, and will need to be flashed in order to load the firmware on. The process involves flashing a runtime image onto an SD card and once it boots, the runtime flashes the board with a system image.

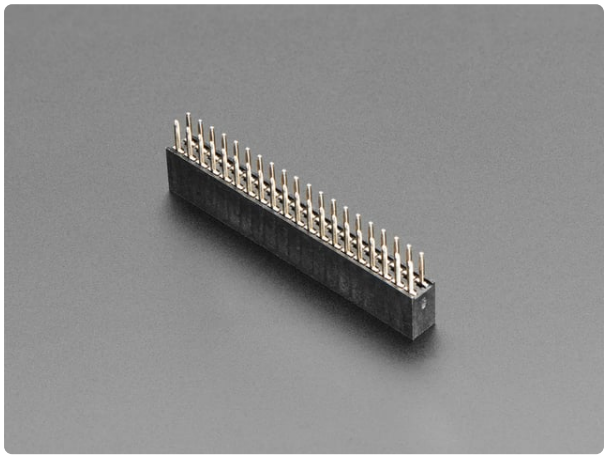
The Coral Getting Started guide will show you the process. (https://adafru.it/ENK) It's not pleasant unless you happen to have all the right cables and software already, it should take you around 15 minutes.

You'll need USB C cables and adapters!

USB 3-in-1 Sync and Charge Cable - Micro B / Type-C / Lightning
As USB technology evolves you'll want the One Cable To Sync/Charge All Things (or, at least, portable devices with Micro-B, Type C, or Lightning ports) and this...
https://www.adafruit.com/product/3679

[USB A to USB C Adapter](https://www.adafruit.com/product/4175)
As technology changes and adapts, so does Adafruit, and speaking of adapting, this adapter has a USB A plug and a USB C socket so your older...
https://www.adafruit.com/product/4175

When connecting HATs or Bonnets, the tall heatsink of the Coral can interfere with attaching it - be sure to pick up some 2x20 GPIO Lifters or Stacking headers!

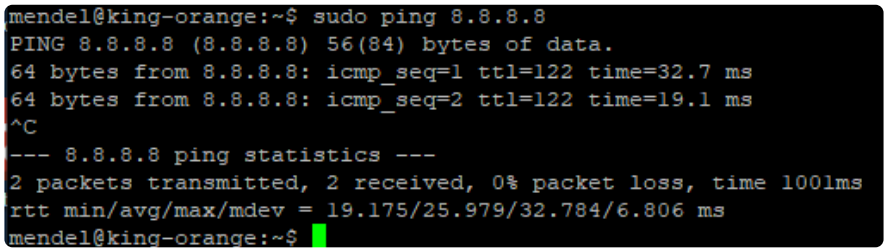


[2x20 Socket Riser Header for Raspberry Pi HATs and Bonnets](https://www.adafruit.com/product/4079)
Give yourself a lift with this 2x20 female socket header that has slightly longer than usual pins. You can plug this into your Raspberry Pi GPIO port and then plug a HAT or...
https://www.adafruit.com/product/4079

[Stacking Header for Pi A+/B+/Pi 2/Pi 3 - 2x20 Extra Tall Header](https://www.adafruit.com/product/1979)
Stack multiple plates, breakouts etc onto your Raspberry Pi Model B+ with this custom-made extra-tall and extra-long 2x20 female header. The female header part has extra spaces to...
https://www.adafruit.com/product/1979

Once you've got the Coral flashed, you will be able to set up and test WiFi:

Verify you have a WiFi connection with **sudo ping 8.8.8.8**



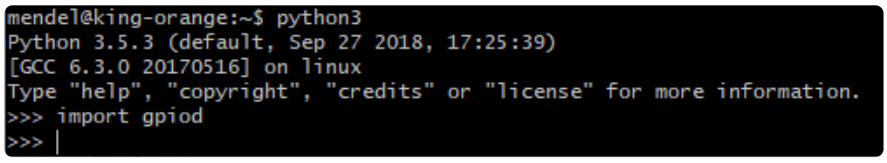
The good news about the **mendel** distribution is it already has Python3 installed by default

Install libgpiod

libgpiod is what we use for gpio toggling. To install run this command:

```
sudo apt-get install libgpiod2
```

After installation you should be able to `import gpiod` from within Python3



Update Your Board and Python

Run the standard updates:

```
sudo apt-get update
sudo apt-get upgrade
```

and

```
sudo pip3 install --upgrade setuptools
```

Check UART, I2C and SPI

A vast number of our CircuitPython drivers use UART, I2C and SPI for interfacing. Luckily, they're already enabled. Check by running `ls /dev/i2c* /dev/spi*`

```
mendel@king-orange:~$ ls /dev/i2c* /dev/spi*
/dev/i2c-0  /dev/i2c-1  /dev/i2c-2  /dev/spidev32766.0  /dev/spidev32766.1
mendel@king-orange:~$
```

Install the support software with

```
sudo apt-get install -y python3-smbus python3-dev i2c-tools
sudo adduser mendel i2c
```

```
mendel@king-orange:~$ sudo apt-get install -y python3-smbus python3-dev i2c-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
i2c-tools is already the newest version (3.1.2-3).
python3-dev is already the newest version (3.5.3-1).
The following NEW packages will be installed:
  python3-smbus
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 9890 B of archives.
After this operation, 32.8 kB of additional disk space will be used.
Get:1 https://packages.cloud.google.com/apt mendel-beaker/main arm64 python3-smbus arm64 3.1.2-3 [9890 B]
Fetched 9890 B in 0s (14.8 kB/s)
Selecting previously unselected package python3-smbus:arm64.
(Reading database ... 43251 files and directories currently installed.)
Preparing to unpack .../python3-smbus_3.1.2-3_arm64.deb ...
Unpacking python3-smbus:arm64 (3.1.2-3) ...
Setting up python3-smbus:arm64 (3.1.2-3) ...
```

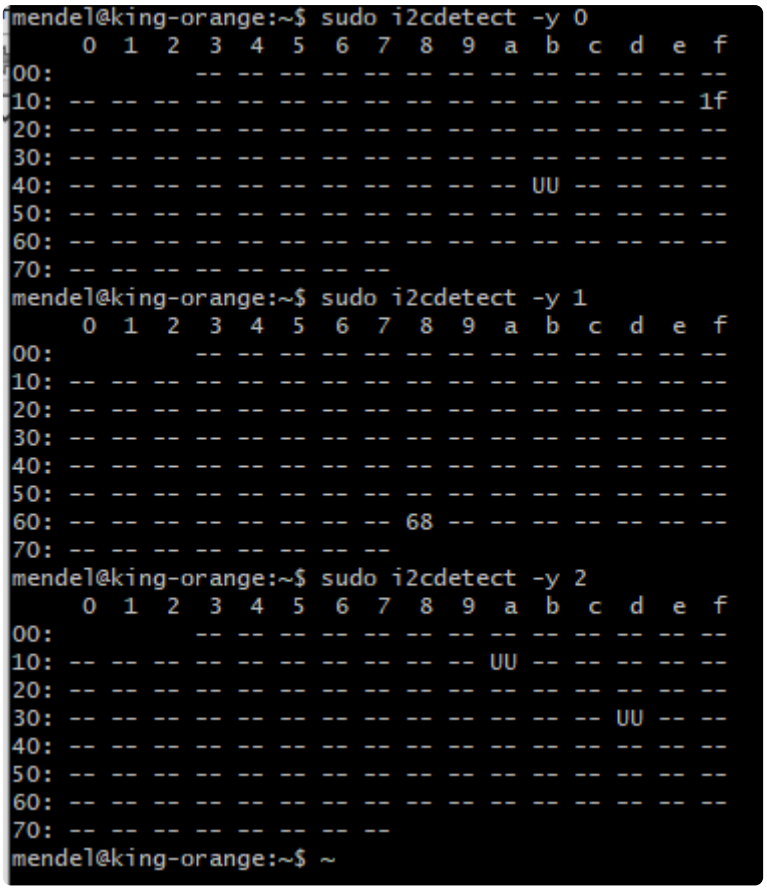
```
mendel@king-orange:~$ sudo adduser mendel i2c
The user `mendel' is already a member of `i2c'.
```

You can get info about the I2C interfaces with **sudo i2cdetect -l**

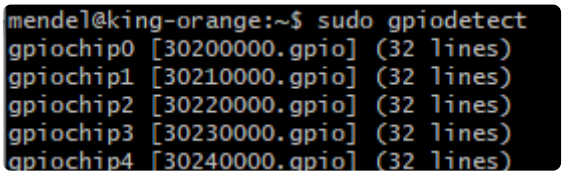
```
mendel@king-orange:~$ sudo i2cdetect -l
i2c-1   i2c             30a30000.i2c                            I2C adapter
i2c-2   i2c             30a40000.i2c                            I2C adapter
i2c-0   i2c             30a20000.i2c                            I2C adapter
```

You can test to see what I2C addresses are connected by running `sudo i2cdetect -y 0` (internal I2C) or `sudo i2cdetect -y 1` (pins #3 and #5) and `sudo i2cdetect -y 1` (pins #27 and #28)

You'll note there looks like an RTC on address 0x68 (a common RTC address). Some addresses are pre-allocated by the kernel (unavailable **UU**)



For the GPIO interface chips, you can check with **sudo gpiodetect** and **sudo gpioinfo**to see the 5 32-pin busses



To enable UART1, you can use the following command:

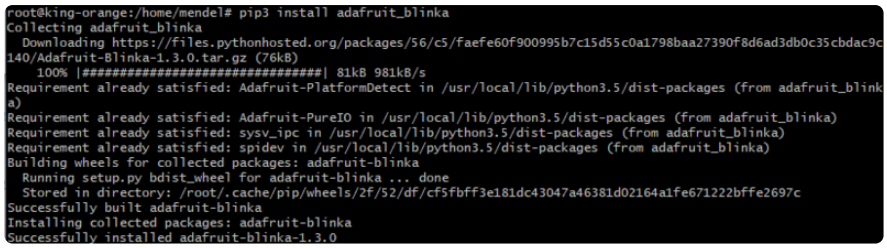
```
systemctl stop serial-getty@ttymxc0.service
```

Install Python libraries

Now you're ready to install all the python support

Run the following command to install **adafruit_blinka**

```
pip3 install adafruit-blinka
```



The computer will install a few different libraries such as `adafruit-pureio` (our ioctl-only i2c library), `spidev` (for SPI interfacing), `Adafruit-GPIO` (for detecting your board) and of course `adafruit-blinka`

That's pretty much it! You're now ready to test.

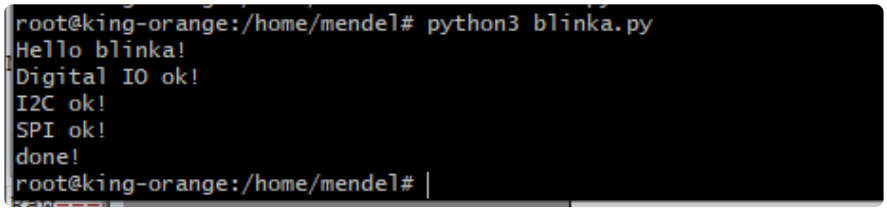
Create a new file called **blinkatest.py** with **nano** or your favorite text editor and put the following in:

```
import board
import digitalio
import busio

print("Hello blinka!")

# Try to great a Digital input
pin = digitalio.DigitalInOut(board.GPIO_P13)
print("Digital IO ok!")

# Try to create an I2C device
i2c = busio.I2C(board.SCL, board.SDA)
print("I2C ok!")

# Try to create an SPI device
spi = busio.SPI(board.SCLK, board.MOSI, board.MISO)
print("SPI ok!")

print("done!")
```

Save it and run at the command line with

```
sudo python3 blinkatest.py
```

You should see the following, indicating digital i/o, I2C and SPI all worked

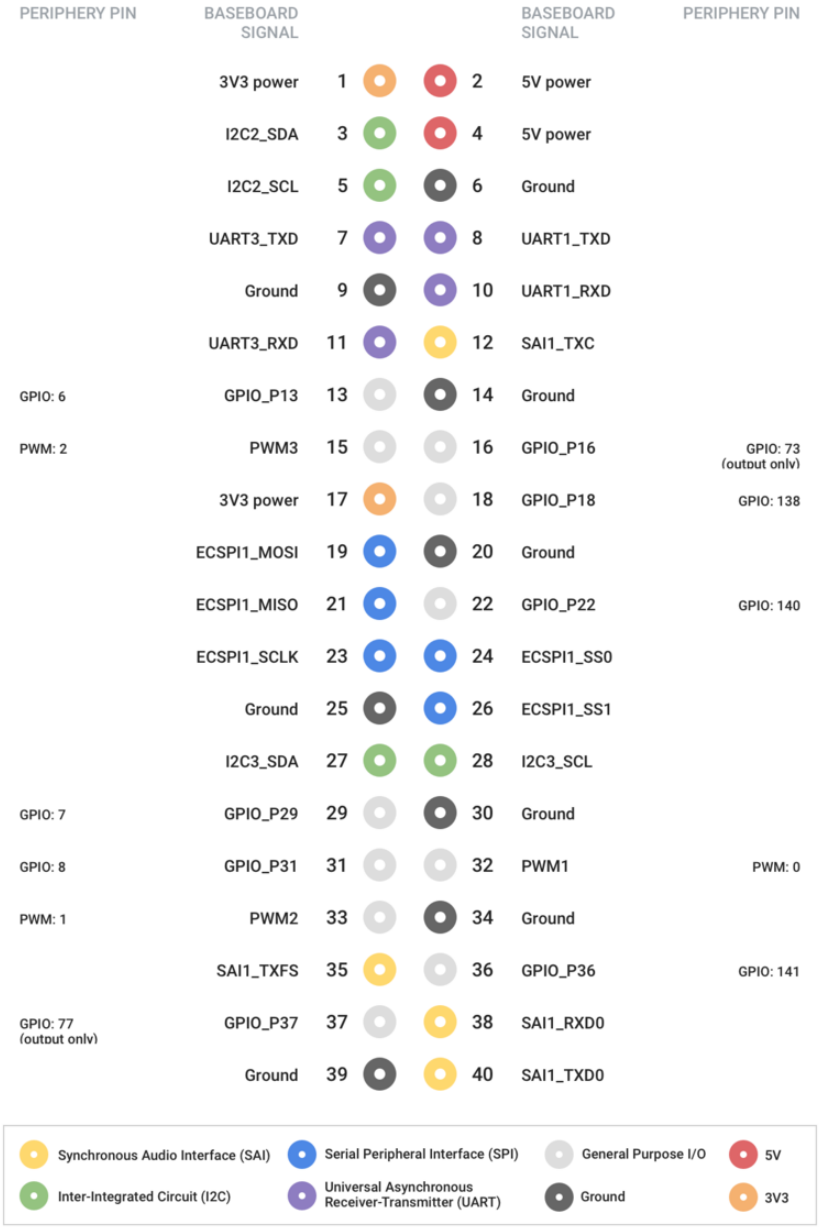


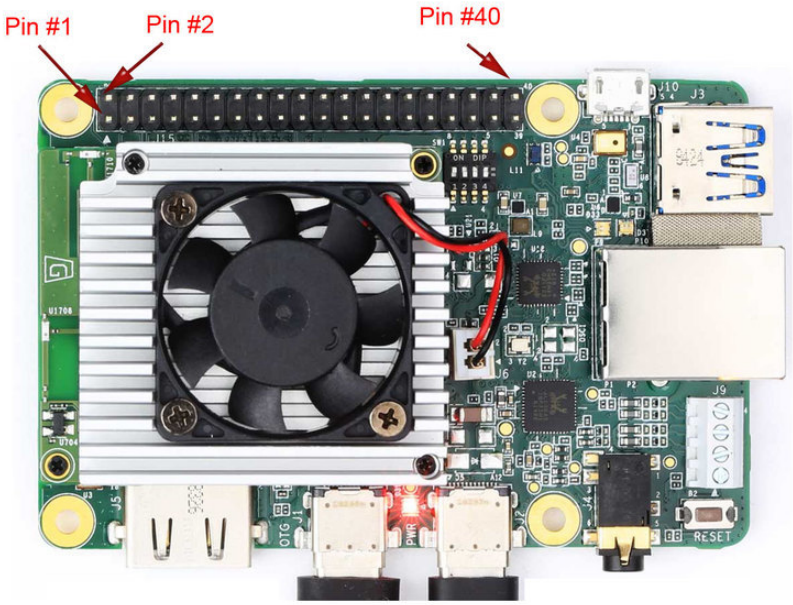
Digital I/O

The first step with any new hardware is the 'hello world' of electronics - blinking an LED. This is very easy with CircuitPython and Coral. We'll extend the example to also show how to wire up a button/switch.

Coral boards don't have any way to set the pullup/pulldown resistors, so you'll need to use external resistors instead of built-in pullups, whenever it makes sense!

Coral Dev Board Pinout





Note that the GPIO pins on the Coral roughly correspond to the Raspberry Pi GPIO

Similarities:

- 5V, 3.3V and GND pins are all in the same locations
- Main I2C port is on pins #3 and #5
- Main SPI port is on pins #19, 21, 23, 24 and 26
- Main UART port is on pins #8 and #10 **note that this is shared with the console so you will conflict with the built in serial console unless you disable the console on these pins**
- I2S is available on same pins as Raspberry Pi

Differences:

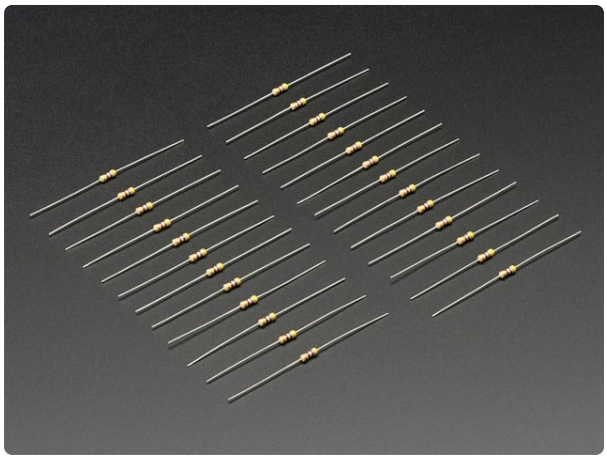
- There's a UART3 on pins #7 and #11 but these don't seem to be available (`/dev/ttymxc2` does not exist) so these pins cannot be used for GPIO
- I2S is enabled by default so you cannot use pins #12, 35, 38, 40 for GPIO
- PWM is enabled on pins #15, #32 and #33 so these pins cannot be used for GPIO but you can use them to create PWM outputs
- The secondary I2C port is available for you to use (pins #27 and #28)
- Raspberry Pi **GPIO #22** is known as **GPIO_P13**
- Raspberry Pi **GPIO #23** is known as **GPIO_P16** (output only)
- Raspberry Pi **GPIO #24** is known as **GPIO_P18**
- Raspberry Pi **GPIO #25** is known as **GPIO_P22**
- Raspberry Pi **GPIO #5** is known as **GPIO_P29**
- Raspberry Pi **GPIO #6** is known as **GPIO_P31**

- Raspberry Pi **GPIO #16** is known as **GPIO_P36**
- Raspberry Pi **GPIO #26** is known as **GPIO_P37** (output only)

Parts Used

Any old LED will work just fine as long as its not an IR LED (you can't see those) and a 470 to 2.2K resistor

Diffused Blue 10mm LED (25 pack)
Need some big indicators? We are big fans of these huge diffused blue LEDs. They are really bright so they can be seen in daytime, and from any angle. They go easily into a breadboard...
https://www.adafruit.com/product/847

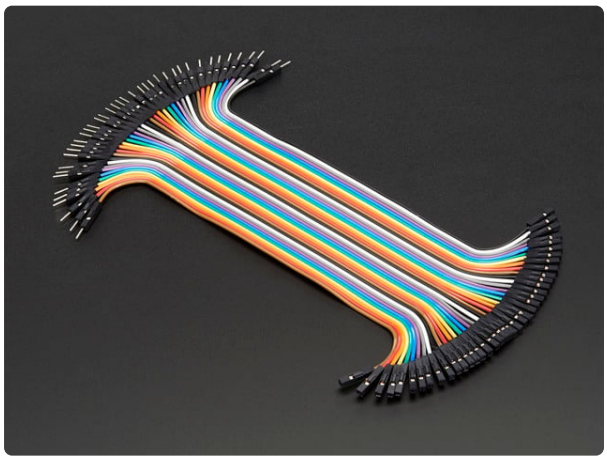


Through-Hole Resistors - 470 ohm 5% 1/4W - Pack of 25
ΩMG! You're not going to be able to resist these handy resistor packs! Well, axially, they do all of the resisting for you!This is a 25 Pack of...
https://www.adafruit.com/product/2781

Some tactile buttons or switches

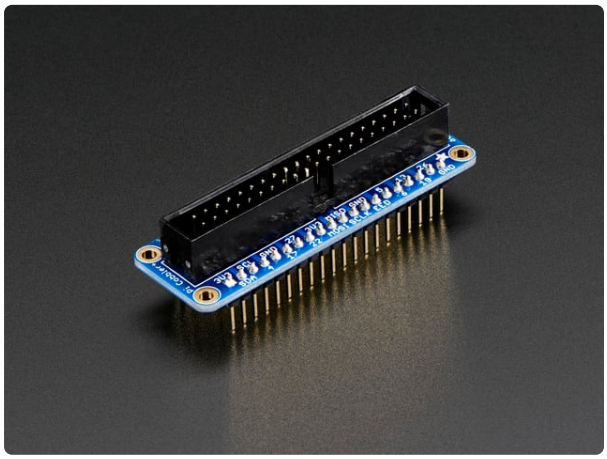
Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack
Medium-sized clicky momentary switches are standard input "buttons" on electronic projects. These work best in a PCB but
https://www.adafruit.com/product/1119

We recommend using a breadboard and some female-male wires.

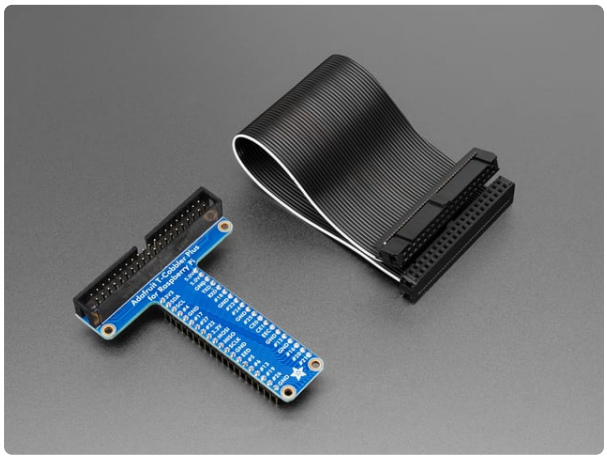


Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)
Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...
https://www.adafruit.com/product/826

You can use a Cobbler to make this a little easier, the pins will be labeled according to Raspberry Pi names so just check the Coral names!



Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3
The Raspberry Pi B+ has landed on the Maker World like a 40-GPIO pinned, quad-USB ported, credit card sized bomb of DIY joy. And while you can use most of our great Model B accessories...
https://www.adafruit.com/product/1990



Assembled Pi T-Cobbler Plus - GPIO Breakout
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...
https://www.adafruit.com/product/2028

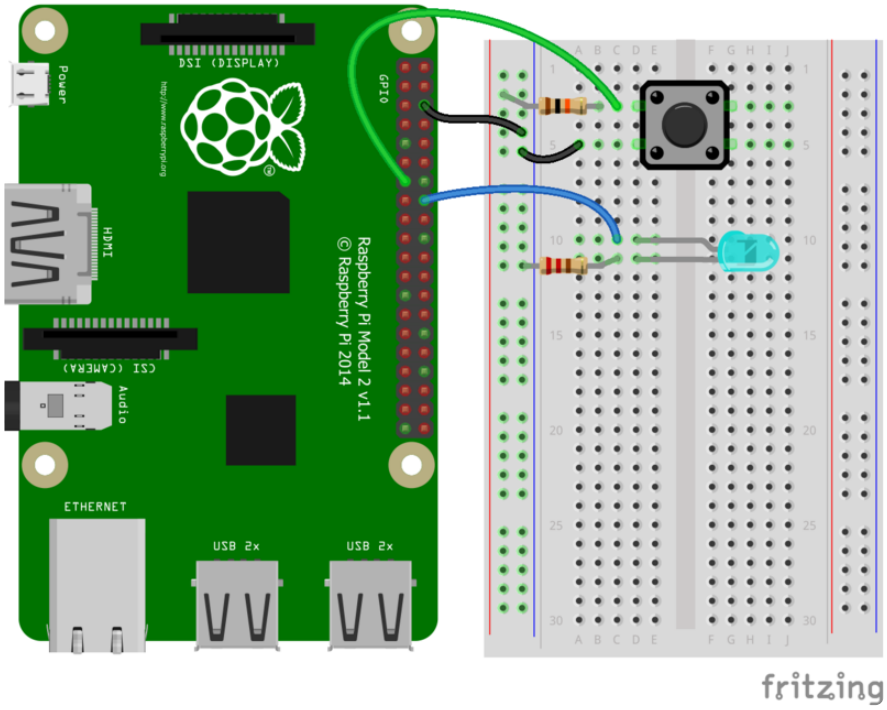
Wiring

Connect the Coral **Ground** pin to the **blue ground rail** on the breadboard.

- Connect one side of the tactile switch to Coral **GPIO_P13**

- Connect a ~10K pull up resistor from **GPIO_P13** to **3.3V**
- Connect the other side of the tactile switch to the **ground** rail
- Connect the longer/positive pin of the LED to Coral **GPIO_P16**
- Connect the shorter/negative pin of the LED to a 470ohm to 2.2K resistor, the other side of the resistor goes to **ground** rail

There's no Coral Fritzing object, so we sub'd a Raspberry Pi in



Double-check you have the right wires connected to the right location, it can be tough to keep track of pins as there are forty of them!

No additional libraries are needed so we can go straight on to the example code

However, we recommend running a pip3 update!

```
pip3 install --upgrade adafruit_blinka
```

Blinky Time!

The finish line is right up ahead, lets start with an example that blinks the LED on and off once a second (half a second on, half a second off):

```
import time
import board
import digitalio

print("hello blinky!")

led = digitalio.DigitalInOut(board.GPIO_P16)
led.direction = digitalio.Direction.OUTPUT
```

```
while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Verify the LED is blinking. If not, check that it's wired to GPIO #16, the resistor is installed correctly, and you have a Ground wire to the Coral.

Type Control-C to quit

Button It Up

Now that you have the LED working, lets add code so the LED turns on whenever the button is pressed

```
import time
import board
import digitalio

print("press the button!")

led = digitalio.DigitalInOut(board.GPIO_P16)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.GPIO_P13)
button.direction = digitalio.Direction.INPUT
# use an external pullup since we don't have internal PU's
#button.pull = digitalio.Pull.UP

while True:
    led.value = not button.value # light when button is pressed!
```

Press the button - see that the LED lights up!

Type Control-C to quit

PWM Outputs & Servos

Some Linux boards, like the Coral, have PWM outputs. You can use these to pulse LEDs to dim them, or color mix, control motor speeds with a motor driver, or even hobby servos!

Supported Pins

The Coral only has 3 PWM pins:

- Pin #33 is **PWM1**
- Pin #32 is **PWM2**
- Pin #15 is **PWM3**

These are independent PWM outputs, each can have a different frequency and duty cycle

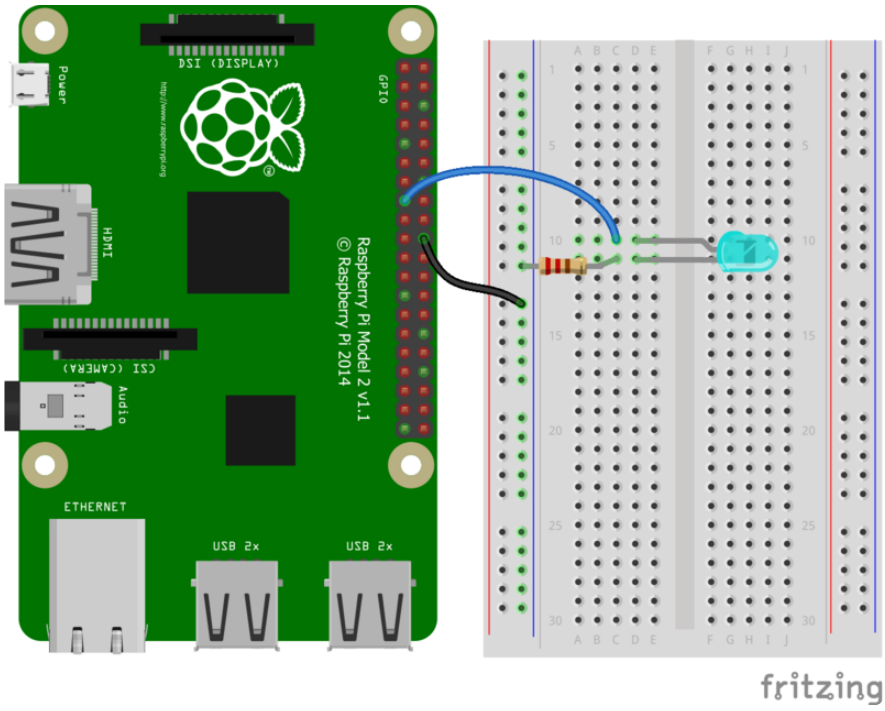
You can check that the 3 PWM's are enabled with `ls /sys/class/pwm/`

```
root@king-orange:/home/mendel# ls /sys/class/pwm/
pwmchip0  pwmchip1  pwmchip2
```

PWM with Fixed Frequency - LEDs

This example will show you how to use PWM to fade an LED.

Wire up an LED like before, but this time to **PWM3** and **GND**



Double-check you have the right wires connected to the right location, it can be tough to keep track of pins as there are forty of them!

No additional libraries are needed so we can go straight on to the example code

However, we recommend running a pip3 update!

```
pip3 install --upgrade adafruit-blinka
```

```
import time
import board
import pwmio

led = pwmio.PWMOut(board.PWM3, frequency=5000, duty_cycle=0)

while True:
    for i in range(100):
        # PWM LED up and down
        if i < 50:
```

```
        led.duty_cycle = int(i * 2 * 65535 / 100)  # Up
    else:
        led.duty_cycle = 65535 - int((i - 50) * 2 * 65535 / 100)  # Down
    time.sleep(0.01)
```

Verify the LED is blinking. If not, check that it's wired to GPIO #15, the resistor is installed correctly, and you have a Ground wire to the Coral.

Type Control-C to quit

Servo Control

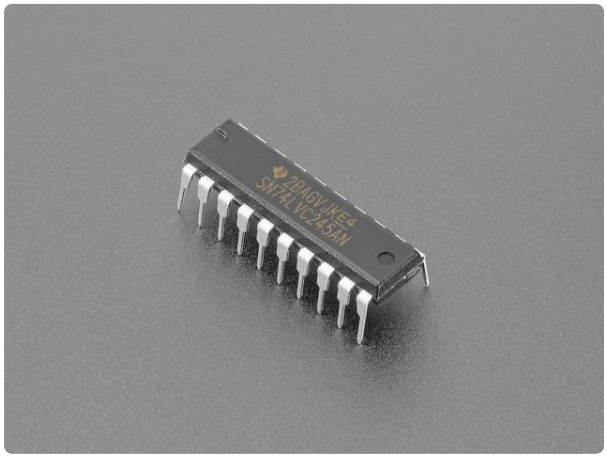
In order to use servos, we take advantage of `pwmio`. Now, in theory, you could just use the raw `pwmio` calls to set the frequency to 50 Hz and then set the pulse widths. But we would rather make it a little more elegant and easy!

So, instead we will use `adafruit_motor` which manages servos for you quite nicely

Install it with `pip3 install adafruit-circuitpython-motor`

The PWM output from the Coral is only 2.5V peak, and not very strong, so you need to buffer it, we recommend using an HC4050 or similar low cost level shifter/buffer. Wire **VCC** and **GND** to **5V** and **GND** on the Coral, and then the PWM output goes into one of the 6 inputs of the '4050.

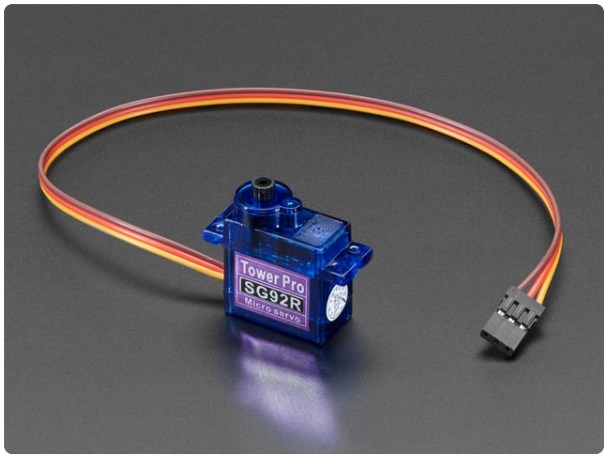
- Connect the servo's **brown** or **black** ground wire to ground
- Connect the servo's **red** power wire to 5V power, USB power is good for a servo or two. For more than that, you'll need an external battery pack. Do not use 3.3V for powering a servo!
- Connect the servo's **yellow** or **white** signal wire to the control/data pin. In this case we're using the shifted **PWM3**



74LVC245 - Breadboard Friendly 8-bit Logic Level Shifter

Most of our customers love using the Arduino for prototyping, design, and invention but find themselves stuck when trying to connect the Arduino to the latest sensors, displays,...

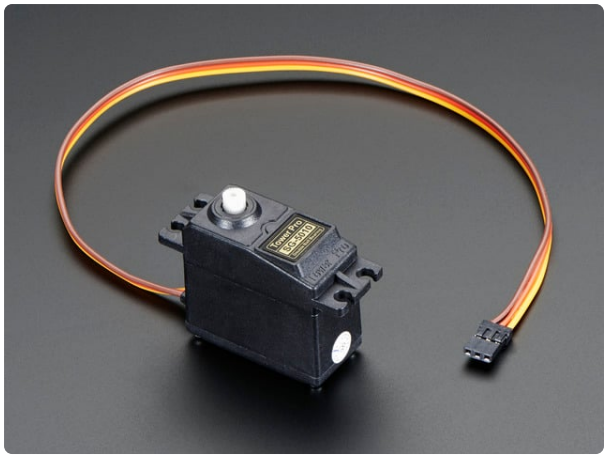
https://www.adafruit.com/product/735



Micro servo

Tiny little servo can rotate approximately 180 degrees (90 in each direction) and works just like the standard kinds you're used to but smaller. You can use any servo...

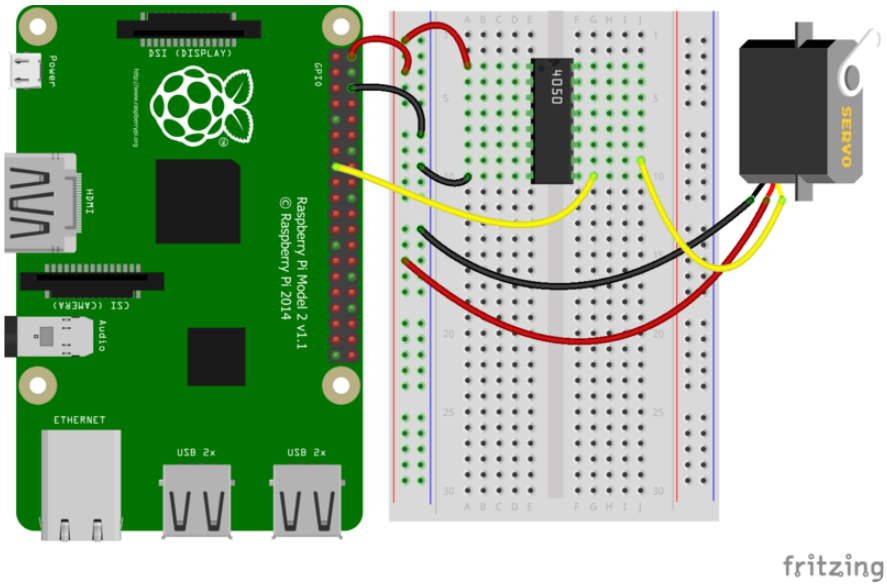
https://www.adafruit.com/product/169



Standard servo - TowerPro SG-5010

This high-torque standard servo can rotate approximately 180 degrees (90 in each direction). You can use any servo code, hardware, or library to control these servos. Good for...

https://www.adafruit.com/product/155



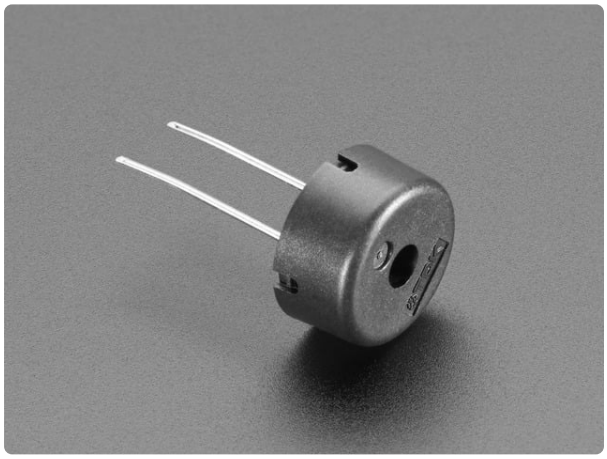

```
import time
import board
import pwmio
from adafruit_motor import servo

# create a PWMOut object on Pin PWM3.
pwm = pwmio.PWMOut(board.PWM3, duty_cycle=2 ** 15, frequency=50)

# Create a servo object, my_servo.
my_servo = servo.Servo(pwm)
```

```
while True:
    for angle in range(0, 180, 5):  # 0 - 180 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
    for angle in range(180, 0, -5): # 180 - 0 degrees, 5 degrees at a time.
        my_servo.angle = angle
        time.sleep(0.05)
```

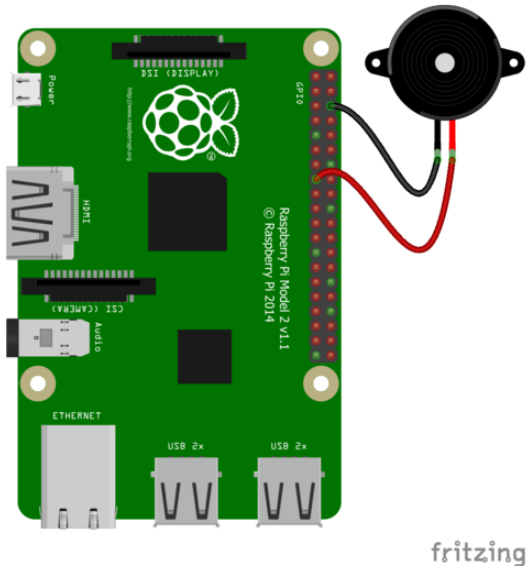
PWM Output with Variable Frequency - Buzzers

This example will show you how to beep a piezo buzzer at different PWM frequencies

Connect a common Piezo buzzer to **PWM3** and **GND**. Don't connect a speaker unless you have an amplifier - the pins on the Coral can only drive small buzzers!



Piezo Buzzer
Piezo buzzers are used for making beeps, tones and alerts. This one is petite but loud! Drive it with 3-30V peak-to-peak square wave. To use, connect one pin to ground (either one) and...
https://www.adafruit.com/product/160




```
import time
import board
import pwmio

piezo = pwmio.PWMOut(board.PWM3, duty_cycle=0, frequency=440,
variable_frequency=True)

while True:
    for f in (262, 294, 330, 349, 392, 440, 494, 523):
        print(f)
```

```
        piezo.frequency = f
        piezo.duty_cycle = 65536 // 2  # On 50%
        time.sleep(0.25)  # On for 1/4 second
        piezo.duty_cycle = 0  # Off
        time.sleep(0.05)  # Pause between notes
    time.sleep(0.5)
```

If you want to make lots of tones, we recommend using the tone helper functions in **simpleio**

Install it with `pip3 install adafruit-circuitpython-simpleio` and then follow this guide page (https://adafru.it/CTk)

The output of the PWM pins isn't very strong, so if you want louder beeps, use the '4050 level shifting arrangement above

---

I2C Sensors & Devices

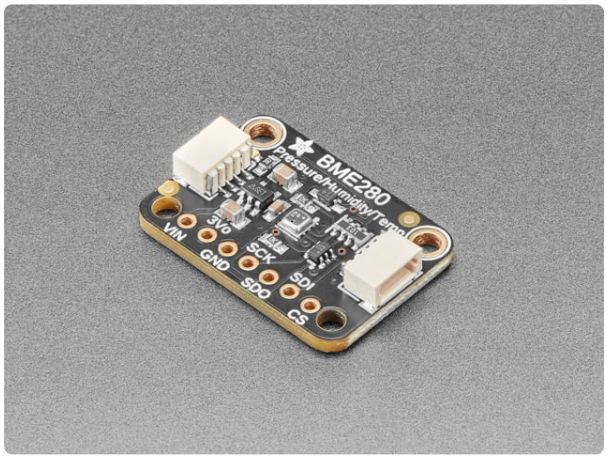
The most popular electronic sensors use I2C to communicate. This is a 'shared bus' 2 wire protocol, you can have multiple sensors connected to the two SDA and SCL pins as long as they have unique addresses (check this guide for a list of many popular devices and their addresses (https://adafru.it/BK0))

Lets show how to wire up a popular BME280. This sensor provides temperature, barometric pressure and humidity data over I2C

We're going to do this in a lot more depth than our guide pages for each sensor, but the overall technique is basically identical for any and all I2C sensors.

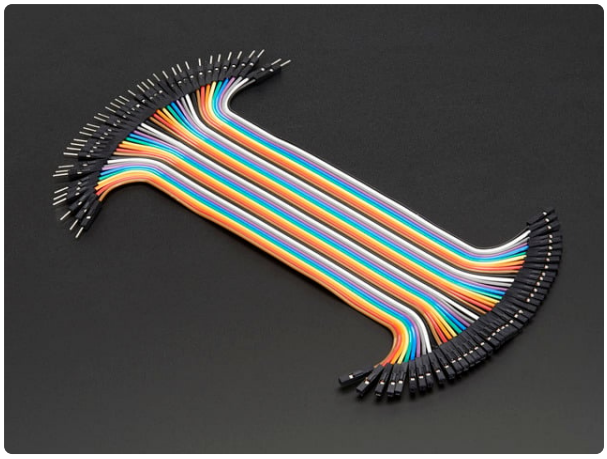
Honestly, the hardest part of using I2C devices is figuring out the I2C address (https://adafru.it/BK0) and which pin is SDA and which pin is SCL!

Parts Used



Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor
Bosch has stepped up their game with their new BME280 sensor, an environmental sensor with temperature, barometric pressure and humidity! This sensor is great for all sorts...
https://www.adafruit.com/product/2652

We recommend using a breadboard and some female-male wires.

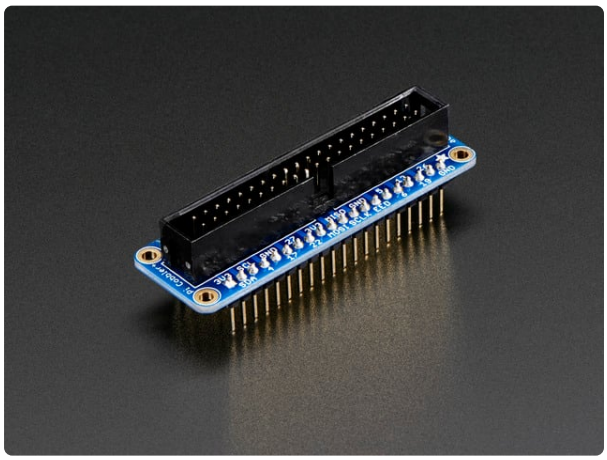


Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)

Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...

https://www.adafruit.com/product/826

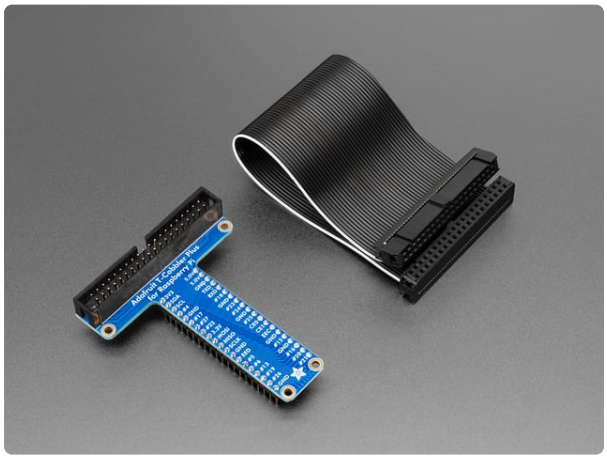
You can use a Cobbler to make this a little easier, the pins are then labeled!



Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3

The Raspberry Pi B+ has landed on the Maker World like a 40-GPIO pinned, quad-USB ported, credit card sized bomb of DIY joy. And while you can use most of our great Model B accessories...

https://www.adafruit.com/product/1990



Assembled Pi T-Cobbler Plus - GPIO Breakout

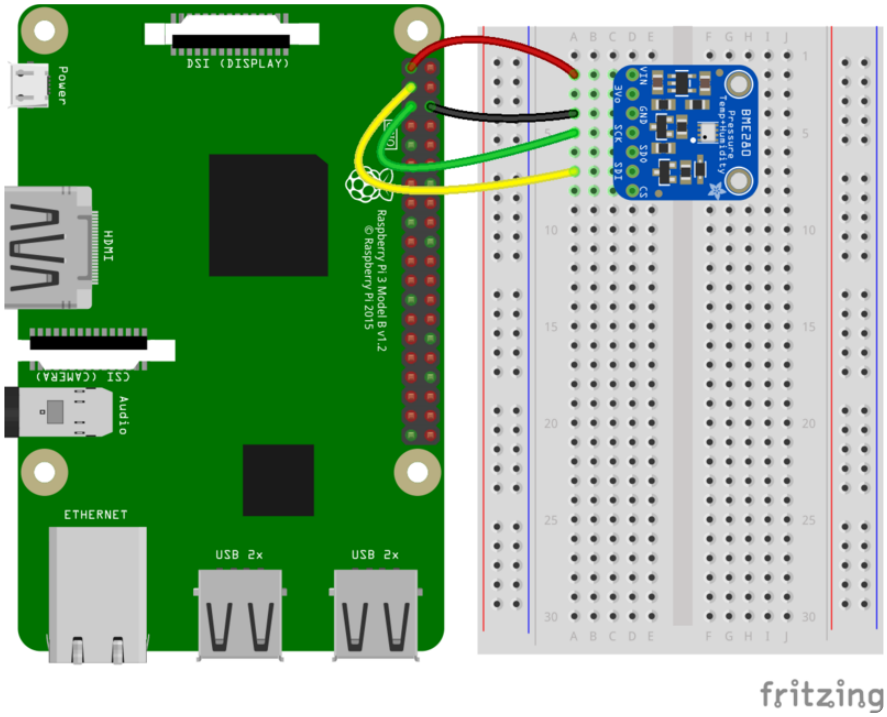
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...

https://www.adafruit.com/product/2028

Wiring

- Connect the Coral **3.3V** power pin to **Vin**
- Connect the Coral **GND** pin to **GND**
- Connect the Pi **SDA** pin to the BME280 **SDI**
- Connect the Pi **SCL** pin to to the BME280 **SCK**

There's no Coral Fritzing object so we're showing Raspberry Pi which has the same pinout



Double-check you have the right wires connected to the right location, it can be tough to keep track of Pi pins as there are forty of them!

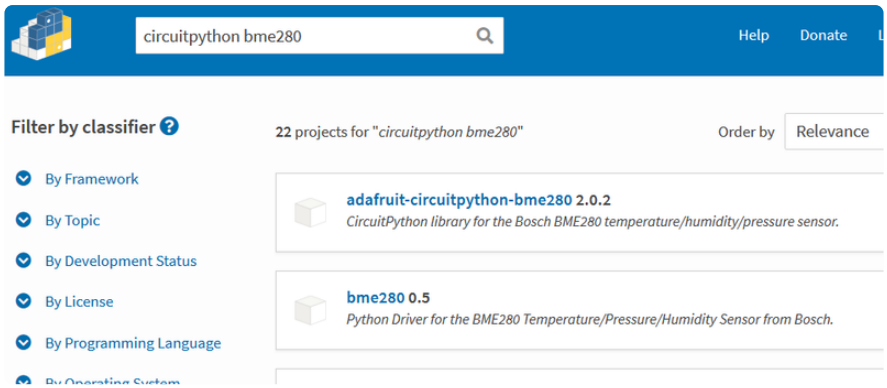
After wiring, we recommend running I2C detection with `sudo i2cdetect -y 1` to verify that you see the device, in this case its address **77**

```
root@king-orange:/home/mendel# sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

Install the CircuitPython BME280 Library

OK onto the good stuff, you can now install the Adafruit BME280 CircuitPython library.

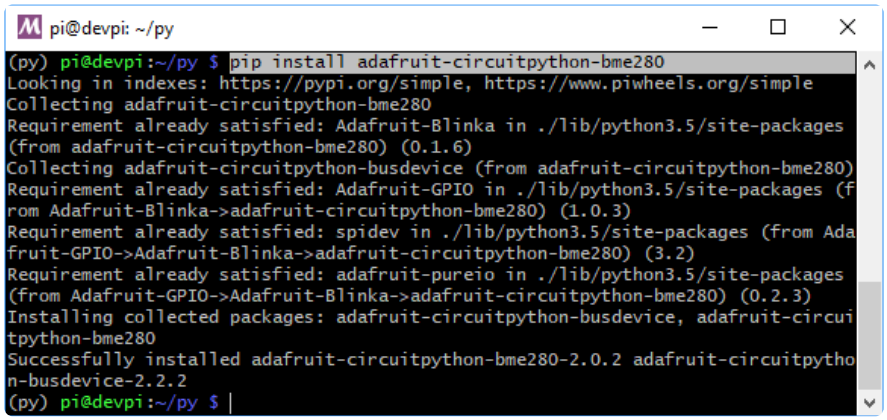
As of this writing, not all libraries are up on PyPI so you may want to search before trying to install. Look for **circuitpython** and then the driver you want.



(If you don't see it [you can open up a github issue on circuitpython to remind us (https://adafru.it/tB7)](https://adafru.it/tB7)!)

Once you know the name, install it with

```
pip3 install adafruit-circuitpython-bme280
```



You'll notice we also installed a dependancy called **adafruit-circuitpython-busdevice**. This is a great thing about pip, if you have other required libraries they'll get installed too!

We also recommend an adafruit-blinka update in case we've fixed bugs:

```
pip3 install --upgrade adafruit_blinka
```

Run that code!

The finish line is right up ahead. You can now run one of the (many in some cases) example scripts we've written for you.

Check out the examples for your library by visiting the repository for the library and looking in the example folder. In this case, it would be [https://github.com/adafruit/Adafruit_CircuitPython_BME280/tree/master/examples (https://adafru.it/BK1)](https://github.com/adafruit/Adafruit_CircuitPython_BME280/tree/master/examples)

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
```

```
import time

import board

from adafruit_bme280 import basic as adafruit_bme280

# Create sensor object, using the board's default I2C bus.
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)

# OR create sensor object, using the board's default SPI bus.
# import digitalio
# spi = board.SPI()
# bme_cs = digitalio.DigitalInOut(board.D10)
# bme280 = adafruit_bme280.Adafruit_BME280_SPI(spi, bme_cs)

# change this to match the location's pressure (hPa) at sea level
bme280.sea_level_pressure = 1013.25

while True:
    print("\nTemperature: %0.1f C" % bme280.temperature)
    print("Humidity: %0.1f %%" % bme280.relative_humidity)
    print("Pressure: %0.1f hPa" % bme280.pressure)
    print("Altitude = %0.2f meters" % bme280.altitude)
    time.sleep(2)
```

Save this code to your Pi by copying and pasting it into a text file, downloading it directly from the Pi, etc.

Then in your command line run

`python3 bme280_simpletest.py`

```
root@king-orange:/home/mendel# python3 bme280_simpletest.py

Temperature: 25.1 C
Humidity: 40.8 %
Pressure: 1008.1 hPa
Altitude = 43.26 meters

Temperature: 25.1 C
Humidity: 40.6 %
Pressure: 1008.1 hPa
Altitude = 43.24 meters

Temperature: 25.1 C
Humidity: 40.4 %
Pressure: 1008.1 hPa
Altitude = 43.19 meters
^CTraceback (most recent call last):
  File "bme280_simpletest.py", line 24, in <module>
    time.sleep(2)
KeyboardInterrupt
root@king-orange:/home/mendel#
```

The code will loop with the sensor data until you quit with a Control-C

That's it! Now if you want to read the documentation on the library, what each function does in depth, visit our readthedocs documentation at

https://circuitpython.readthedocs.io/projects/bme280/en/latest/ (https://adafru.it/BK2)

SPI Sensors & Devices

SPI is less popular than I2C but still you'll see lots of sensors and chips use it. Unlike I2C, you don't have everything share two wires. Instead, there's three shared wires (**clock**, **data in**, **data out**) and then a unique '**chip select**' line for each chip.

The nice thing about SPI is you can have as many chips as you like, even the same kind, all share the three SPI wires, as long as each one has a unique chip select pin.

The formal/technical names for the 4 pins used are:

- SPI clock - called **SCLK**, **SCK** or **CLK**
- SPI data out - called **MOSI** for **M**icrocomputer **O**ut **S**erial **I**n. This is the wire that takes data from the Linux computer to the sensor/chip. Sometimes marked **SDI** or **DI** on chips
- SPI data in - called **MISO** for **M**icrocomputer **I**n **S**erial **O**ut. This is the wire that takes data to the Linux computer from the sensor/chip. Sometimes marked **SDO** or **DO** on chips
- SPI chip select - called **CS** or **CE** or **SS**

Remember, connect all SCK, MOSI and MISO pins together (unless there's some specific reason/instruction not to) and a unique CS pin for each device.

WARNING! SPI on Linux/Coral WARNING!

SPI on microcontrollers is fairly simple, you have an SPI peripheral and you can transfer data on it with some low level command. Its 'your job' as a programmer to control the CS lines with a GPIO. That's how CircuitPython is structured as well. `busio` does just the SPI transmit/receive part and `busdevice` handles the chip select pin as well.

Linux, on the other hand, doesn't let you send data to SPI without a CS line, and the CS lines are fixed in hardware as well. For example on the Coral, there's two CS pins available for the hardware SPI pins - **ESPI1_SS0** and **ESPI1_SS1** - and you have to use them. (In theory there's an ioctl option called `no_cs` but this does not actually work)

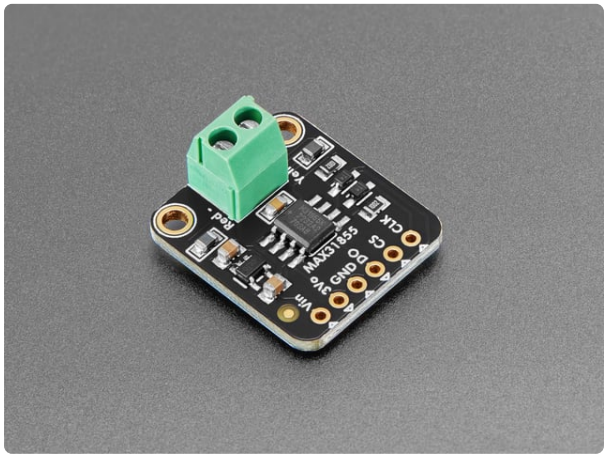
The upshot here is - to let you use more than 1 peripheral on SPI, we decided to **let you use any CS pins** you like, CircuitPython will toggle it the way you expect. But

when we transfer SPI data we always tell the kernel to use **ESPI1_SS0**. **ESPI1_SS0** will toggle like a CS pin, but if we leave it disconnected, its no big deal

The upshot here is basically never connect anything to ESPI1_SS0. Use whatever chip select pin you define in CircuitPython and just leave the **ESPI1_SS0** pin alone, it will toggle as if it is the chip select line, completely on its own, so you shouldn't try to use it as a digital input/output/whatever.

Parts Used

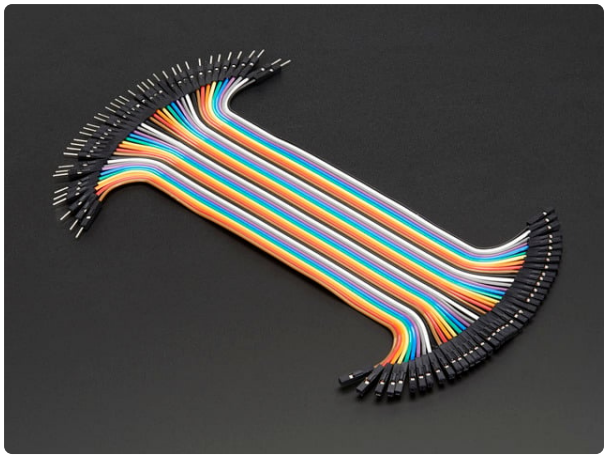
OK now that we've gone thru the warning, lets wire up an SPI MAX31855 thermocouple sensor, this particular device doesn't have a MOSI pin so we'll not connect it.



Thermocouple Amplifier MAX31855 breakout board (MAX6675 upgrade)
Thermocouples are very sensitive, requiring a good amplifier with a cold-compensation reference. The MAX31855K does everything for you, and can be easily interfaced with any...
https://www.adafruit.com/product/269

Thermocouple Type-K Glass Braid Insulated
Thermocouples are best used for measuring temperatures that can go above 100 °C. This is a bare wires bead-probe which can measure air or surface temperatures. Most inexpensive...
https://www.adafruit.com/product/270

We recommend using a breadboard and some female-male wires.

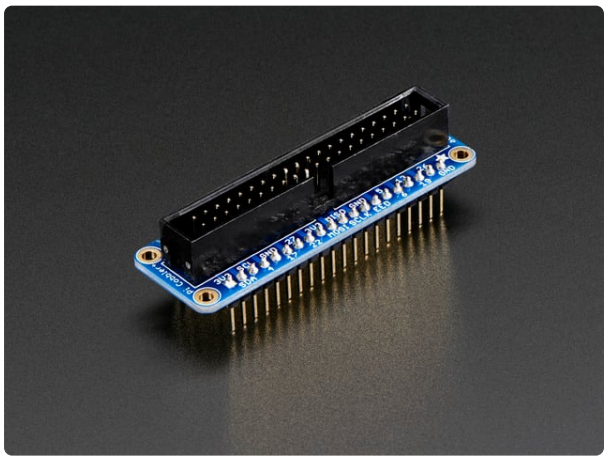


Premium Female/Male 'Extension' Jumper Wires - 40 x 6" (150mm)

Handy for making wire harnesses or jumpering between headers on PCB's. These premium jumper wires are 6" (150mm) long and come in a 'strip' of 40 (4 pieces of each of...

https://www.adafruit.com/product/826

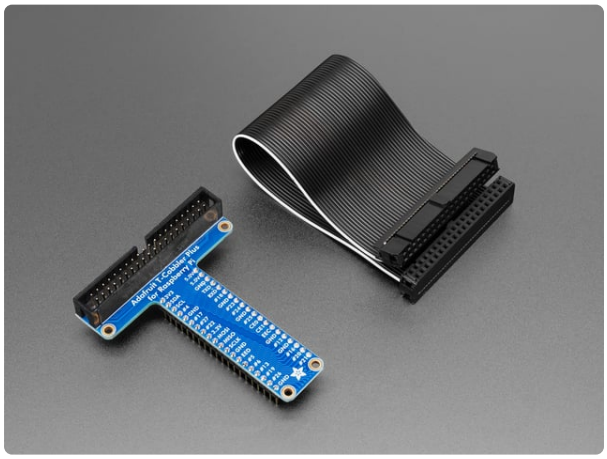
You can use a Cobbler to make this a little easier, the pins are then labeled!



Adafruit Pi Cobbler + Kit- Breakout Cable for Pi B+/A+/Pi 2/Pi 3

The Raspberry Pi B+ has landed on the Maker World like a 40-GPIO pinned, quad-USB ported, credit card sized bomb of DIY joy. And while you can use most of our great Model B accessories...

https://www.adafruit.com/product/1990



Assembled Pi T-Cobbler Plus - GPIO Breakout

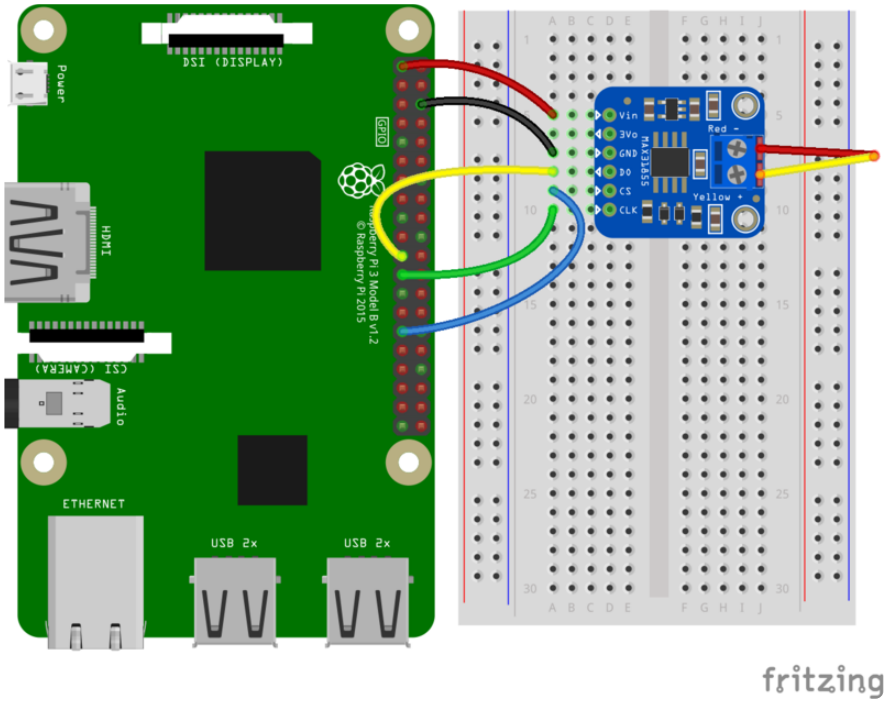
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...

https://www.adafruit.com/product/2028

Wiring

- Connect the Coral **3.3V** power pin to **Vin**
- Connect the Coral **GND** pin to **GND**
- Connect the Coral **SCLK** pin to the MAX31855 **CLK**
- Connect the Coral **MISO** pin to to the MAX31855 **DO**
- Connect the Coral **GPIO_P29** pin to to the MAX31855 **CS**

There's no Coral Fritzing object, so we'll show using a Raspberry Pi which has the same pinout



Double-check you have the right wires connected to the right location, it can be tough to keep track of Pi pins as there are forty of them!

Install the CircuitPython MAX31855 Library

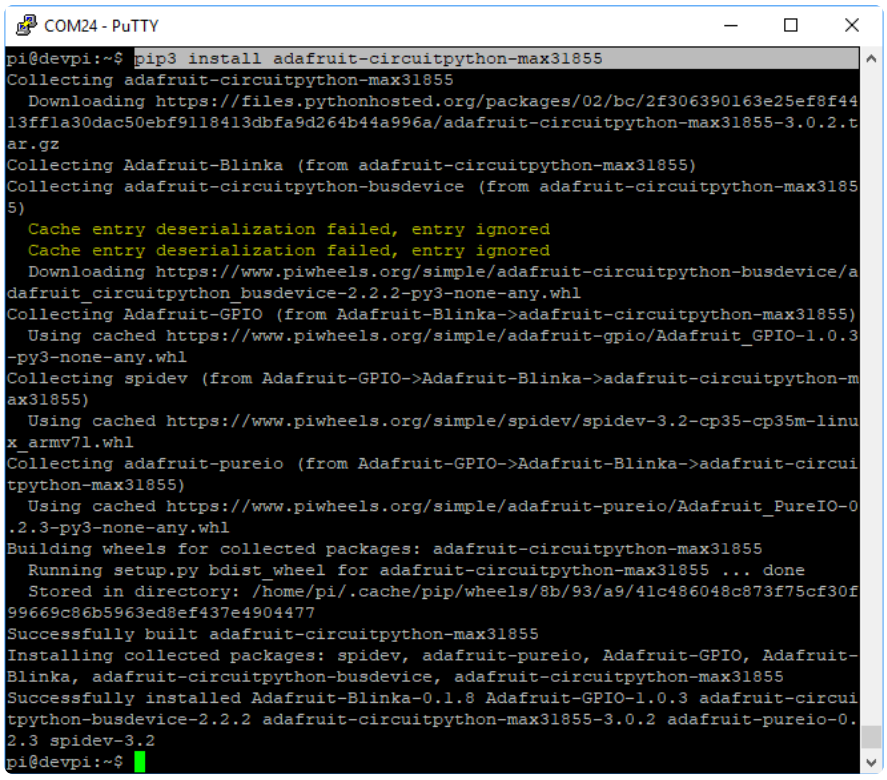
OK onto the good stuff, you can now install the Adafruit MAX31855 CircuitPython library.

As of this writing, not all libraries are up on PyPI so you may want to search before trying to install. Look for **circuitpython** and then the driver you want.

(If you don't see it [you can open up a github issue on circuitpython to remind us (https://adafru.it/tB7)](https://adafru.it/tB7)!)

Once you know the name, install it with

```
pip3 install adafruit-circuitpython-max31855
```



You'll notice we also installed a few other dependancies called **spidev**, **adafruit-pureio**, **adafruit-circuitpython-busdevice** and more. This is a great thing about pip, if you have other required libraries they'll get installed too!

We also recommend an adafruit-blinka update in case we've fixed bugs:

```
pip3 install --upgrade adafruit_blinka
```

Run that code!

The finish line is right up ahead. You can now run one of the (many in some cases) example scripts we've written for you.

Check out the examples for your library by visiting the repository for the library and looking in the example folder. In this case, it would be https://github.com/adafruit/Adafruit_CircuitPython_MAX31855/tree/master/examples (https://adafru.it/BKj)

As of this writing there's only one example. But that's cool, here it is:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time

import board
```

```
import digitalio

import adafruit_max31855

spi = board.SPI()
cs = digitalio.DigitalInOut(board.D5)

max31855 = adafruit_max31855.MAX31855(spi, cs)
while True:
    tempC = max31855.temperature
    tempF = tempC * 9 / 5 + 32
    print(f"Temperature: {tempC} C {tempF} F ")
    time.sleep(2.0)
```

Save this code to your Coral by copying and pasting it into a text file, downloading it directly from the Coral, etc.

Change the line that says

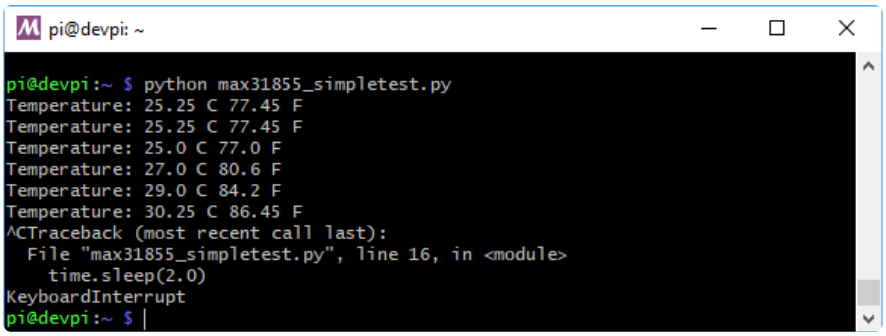
`cs = digitalio.DigitalInOut(board.D5)`

to

`cs = digitalio.DigitalInOut(board.GPIO_P29)`

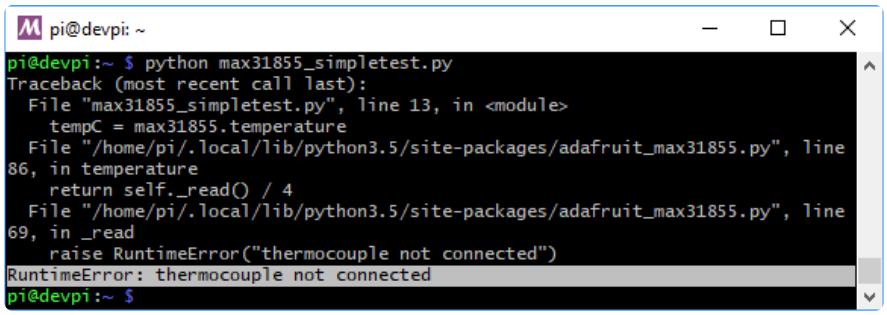
Then in your command line run

`python3 max31855_simpletest.py`



The code will loop with the sensor data until you quit with a Control-C

Make sure you have a K-type thermocouple installed into the sensor breakout or you will get an error like the one below!



That's it! Now if you want to read the documentation on the library, what each function does in depth, visit our readthedocs documentation at

https://circuitpython.readthedocs.io/projects/max31855/en/latest/ (https://adafru.it/BKk)

UART / Serial

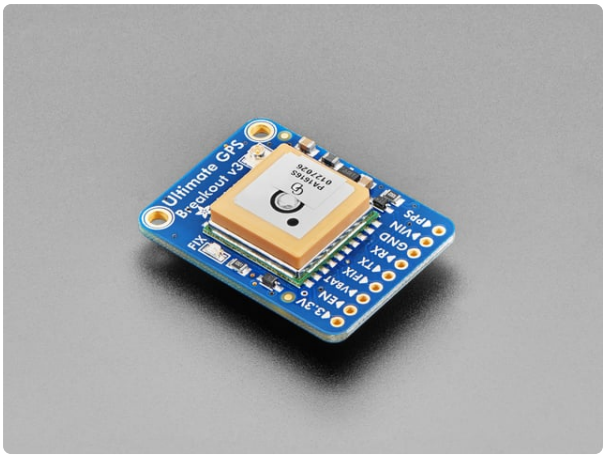
After I2C and SPI, the third most popular "bus" protocol used is serial (also sometimes referred to as 'UART'). This is a non-shared two-wire protocol with an RX line, a TX line and a fixed baudrate. The most common devices that use UART are GPS units, MIDI interfaces, fingerprint sensors, thermal printers, and a scattering of sensors.

One thing you'll notice fast is that most linux computers have minimal UARTs, often only 1 hardware port. And that hardware port may be shared with a console.

There are two ways to connect UART / Serial devices to your Coral. With a USB adapter and via the onboard pins

We'll demonstrate wiring up & using an Ultimate GPS with both methods

> The Coral pinout indicates there's a second UART on two pins but they are not enabled and there's no documentation on how to do that yet, see https://stackoverflow.com/questions/55827855/cannot-open-uart-on-the-40-pin-header



Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates

We carry a few different GPS modules here in the Adafruit shop, but none that satisfied our every desire - that's why we designed this little GPS breakout board. We believe this is...

https://www.adafruit.com/product/746

The Easy Way - An External USB-Serial Converter

By far the easiest way to add a serial port is to use a USB to serial converter cable or breakout. They're not expensive, and you simply plug it into the USB port. On the other end are wires or pins that provide power, ground, RX, TX and maybe some other control pads or extras.

Here are some options, they have varying chipsets and physical designs but all will do the job. We'll list them in order of recommendation.

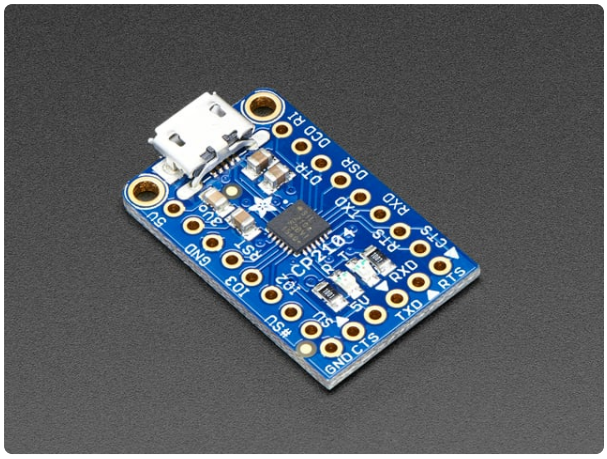
The first cable is easy to use and even has little plugs that you can arrange however you like, it contains a CP2102

USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi

The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...

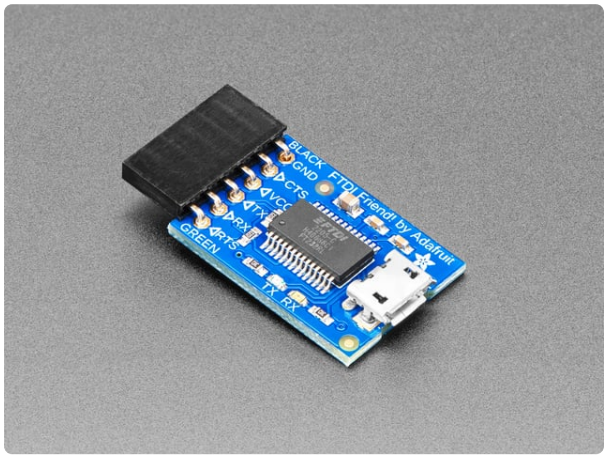
https://www.adafruit.com/product/954

The CP2104 Friend is low cost, easy to use, but requires a little soldering, it has an '6-pin FTDI compatible' connector on the end, but all pins are broken out the sides



[Adafruit CP2104 Friend - USB to Serial Converter]
Discontinued - you can grab Adafruit CP2102N Friend - USB to Serial Converter instead! Long gone are...
https://www.adafruit.com/product/3309

Both the FTDI friend and cable use classic FTDI chips, these are more expensive than the CP2104 or PL2303 but sometimes people like them!

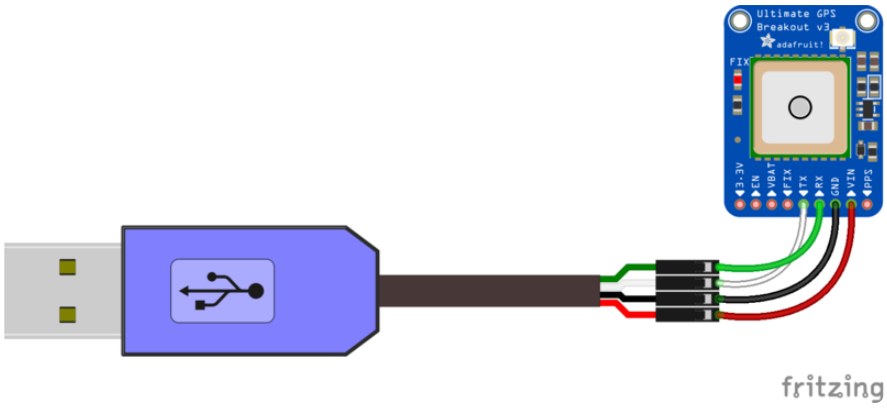


[FTDI Friend with Micro USB Port + extras]
Long gone are the days of parallel ports and serial ports. Now the USB port reigns supreme! But USB is hard, and you just want to transfer your every-day serial data from a...
https://www.adafruit.com/product/284

[FTDI Serial TTL-232 USB Cable]
Just about all electronics use TTL serial for debugging, bootloading, programming, serial output, etc. But it's rare for a computer to have a serial port anymore. This is a USB to...
https://www.adafruit.com/product/70

You can wire up the GPS by connecting the following

- **GPS Vin** to **USB 5V** or **3V** (red wire on USB console cable)
- **GPS Ground** to **USB Ground** (black wire)
- **GPS RX** to **USB TX** (green wire)
- **GPS TX** to **USB RX** (white wire)



Once the USB adapter is plugged in, you'll need to figure out what the serial port name is. You can figure it out by unplugging-replugging in the USB and then typing `dmesg | tail -10` (or just `dmesg`) and looking for text like this:

```
pi@devpi:~ $ dmesg | tail -10
[  601.391424] usb 1-1.2: Manufacturer: FTDI
[  601.391432] usb 1-1.2: SerialNumber: AL00FP25
[  601.440489] usbcore: registered new interface driver usbserial
[  601.440554] usbcore: registered new interface driver usbserial_generic
[  601.440609] usbserial: USB Serial support registered for generic
[  601.455895] usbcore: registered new interface driver ftdi_sio
[  601.455970] usbserial: USB Serial support registered for FTDI USB Serial Device
[  601.456248] ftdi_sio 1-1.2:1.0: FTDI USB Serial Device converter detected
[  601.456383] usb 1-1.2: Detected FT232RL
[  601.459259] usb 1-1.2: FTDI USB Serial Device converter now attached to ttyUSB0
```

At the bottom, you'll see the 'name' of the attached device, in this case its `ttyUSB0`, that means our serial port device is available at `/dev/ttyUSB0`

The Hard Way - Using Built-in UART

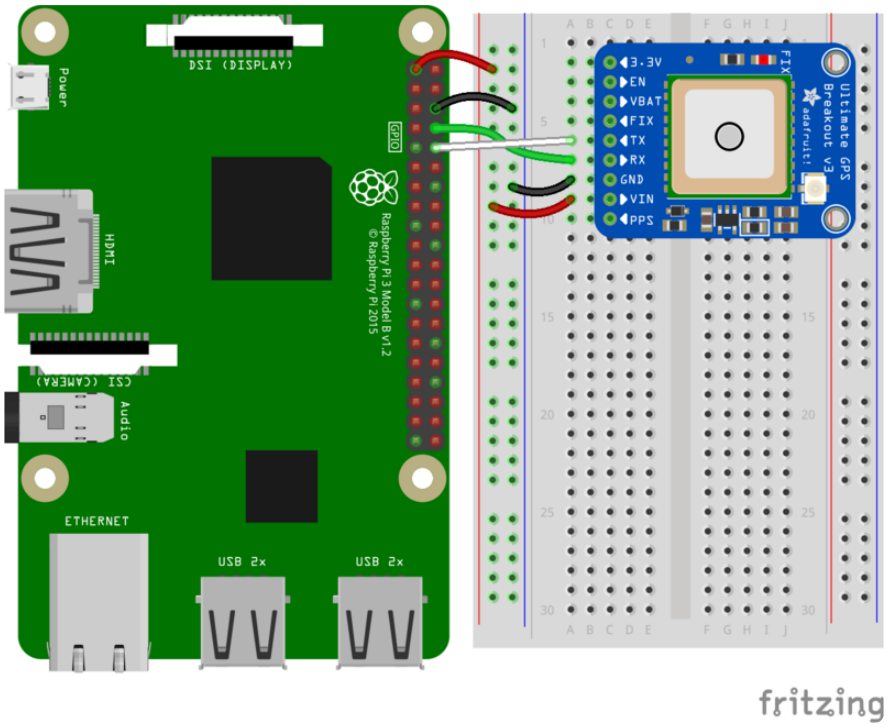
If you don't want to plug in external hardware to the Coral you can use the built in UART on the RX/TX pins.

Like the Raspberry Pi the Coral uses the RX/TX pins for a console, so this isn't recommended because you will conflict with the built in UART console!

You can use the built in UART via `/dev/ttymxc0`

Wire the GPS as follows:

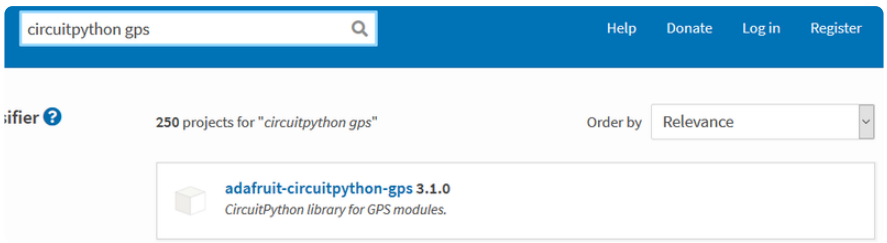
> There's no Coral Fritzing object, but the Raspberry Pi has the same pinout so we're using that instead



Install the CircuitPython GPS Library

OK onto the good stuff, you can now install the Adafruit GPS CircuitPython library.

As of this writing, not all libraries are up on PyPI so you may want to search before trying to install. Look for **circuitpython** and then the driver you want.



(If you don't see it [you can open up a github issue on circuitpython to remind us (https://adafru.it/tB7)](https://adafru.it/tB7)!)

Once you know the name, install it with

```
pip3 install pyserial adafruit-circuitpython-gps
```

You'll notice we also installed a dependancy called **pyserial**. This is a great thing about pip, if you have other required libraries they'll get installed too!

We also recommend an adafruit-blinka update in case we've fixed bugs:

```
pip3 install --upgrade adafruit_blinka
```

Run that code!

The finish line is right up ahead. You can now run one of the (many in some cases) example scripts we've written for you.

Check out the examples for your library by visiting the repository for the library and looking in the example folder. In this case, it would be https://github.com/adafruit/Adafruit_CircuitPython_GPS/tree/master/examples (https://adafru.it/Ca9)

Lets start with the simplest, the echo example

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple GPS module demonstration.
# Will print NMEA sentences received from the GPS, great for testing connection
# Uses the GPS to send some commands, then reads directly from the GPS
import time

import board
import busio

import adafruit_gps

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# for a computer, use the pyserial library for uart access
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
# i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart)  # Use UART/pyserial
# gps = adafruit_gps.GPS_GtopI2C(i2c)  # Use I2C interface

# Initialize the GPS module by changing what data it sends and at what rate.
# These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
# PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
# the GPS module behavior:
#   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

# Turn on the basic GGA and RMC info (what you typically want)
gps.send_command(b"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
# Turn on just minimum info (RMC only, location):
# gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
# gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Tuen on everything (not all of it is parsed!)
# gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

# Set update rate to once a second (1hz) which is what you typically want.
gps.send_command(b"PMTK220,1000")
# Or decrease to once every two seconds by doubling the millisecond value.
# Be sure to also increase your UART timeout above!
# gps.send_command(b'PMTK220,2000')
```

```
# You can also speed up the rate, but don't go too fast or else you can lose
# data during parsing.  This would be twice a second (2hz, 500ms delay):
# gps.send_command(b'PMTK220,500')

# Main loop runs forever printing data as it comes in
timestamp = time.monotonic()
while True:
    data = gps.read(32)  # read up to 32 bytes
    # print(data)  # this is a bytearray type

    if data is not None:
        # convert bytearray to string
        data_string = "".join([chr(b) for b in data])
        print(data_string, end="")

    if time.monotonic() - timestamp > 5:
        # every 5 seconds...
        gps.send_command(b"PMTK605")  # request firmware version
        timestamp = time.monotonic()
```

We'll need to configure this code to work with our UART port name.

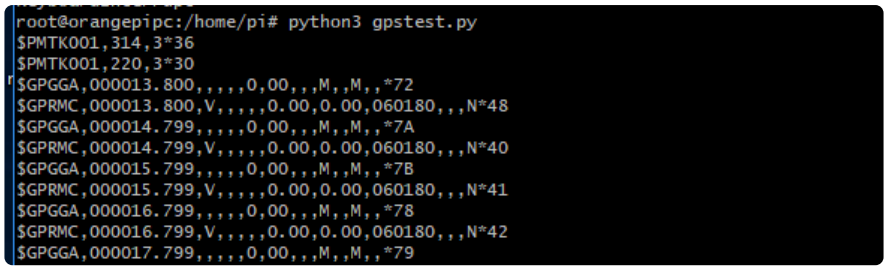
- If you're using a USB-to-serial converter, the device name is probably `/dev/ttyUSB0` - but check `dmesg` to make sure
- If you're using the built-in UART on the Coral, the device name is `/dev/ttymxc0`

Comment out the lines that reference `board.TX`, `board.RX` and `busio.uart` and uncomment the lines that `import serial` and define the `serial` device, like so:

```
# Define RX and TX pins for the board's serial port connected to the GPS.
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
#RX = board.RX
#TX = board.TX

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
#uart = busio.UART(TX, RX, baudrate=9600, timeout=3)

# for a computer, use the pyserial library for uart access
import serial
uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=3)
```

And update the `"/dev/ttyUSB0"` device name if necessary to match your USB interface

Whichever method you use, you should see output like this, with **$GP** "NMEA sentences" - there probably wont be actual location data because you haven't gotten a GPS fix. As long as you see those **$GP** strings sorta like the below, you've got it working!



More To Come!

That's just a taste of what we've got working so far

We're adding more support constantly, so please hold tight and visit the adafruit_blinka github repo (https://adafru.it/BJX) to share your feedback and perhaps even submit some improvements!

ToDo's

- There's no documentation on how to enable UART3 - waiting on Google to publish info
- There's no documentation on how to disable the console to free up UART1 - waiting on Google to publish info

---

FAQ & Troubleshooting

There's a few oddities when running Blinka/CircuitPython on Linux. Here's a list of stuff to watch for that we know of!

This FAQ covers all the various platforms and hardware setups you can run Blinka on. Therefore, some of the information may not apply to your specific setup.

Update Blinka/Platform Libraries

Most issues can be solved by forcing Python to upgrade to the latest `blinka` / `platform-detect` libraries. Try running

```
sudo python3 -m pip install --upgrade --force-reinstall adafruit-blinka Adafruit-PlatformDetect
```

Getting an error message about 'board' not found or 'board' has no attribute

Somehow you have ended up with either the wrong **board** module or no **board** module at all.

DO NOT try to fix this by manually installing a library named `board`. There is one out there (https://adafru.it/NCE) and it has nothing to do with Blinka. You will break things if you install that library!

The easiest way to recover is to simply force a reinstall of Blinka with: `python3 -m pip install --upgrade --force-reinstall adafruit-blinka`

Additionally, and especially if you are using a more recent version of Python, you may run into this error if you either do not have a Virtual Environment active or setup. See the Python Virtual Environment Usage on Raspberry Pi (https://adafru.it/19a5) guide for more information or check out the guide's Installation page.

Mixed SPI mode devices

Due to the way we share an SPI peripheral, you cannot have two SPI devices with different 'mode/polarity' on the same SPI bus - you'll get weird data

95% of SPI devices are mode 0, check the driver to see mode or polarity settings. For example:

- LSM9DS1 is mode 1 (https://adafru.it/NCF), please use in I2C mode instead of SPI
- MAX31865 is phase 1 (https://adafru.it/NCG), try using this on a separate SPI device, or read data twice.

Why am I getting AttributeError: 'SpiDev' object has no attribute 'writebytes2'?

This is due to having an older version of spidev (https://adafru.it/JEi). You need at least version 3.4. This should have been taken care of (https://adafru.it/NCH) when you installed Blinka, but in some cases it does not seem to happen.

To check what version of spidev Python is using:

```
$ python3
Python 3.6.8 (default, Oct 7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>> import spidev
>>> spidev.__version__
'3.4'
>>>
```

If you see a version lower then 3.4 reported, then try a force upgrade of spidev with (back at command line):

```
sudo python3 -m pip install --upgrade --force-reinstall
spidev
```

No Pullup/Pulldown support on some Linux boards or MCP2221

Some Linux boards, for example, AllWinner-based, do not have support to set pull up or pull down on their GPIO. Use an external resistor instead!

Getting OSError: read error with MCP2221

If you are getting a stack trace that ends with something like:

```
return self._hid.read(64)
File "hid.pyx", line 122, in hid.device.read
OSError: read error
```

Try setting an environment variable named **BLINKA_MCP2221_RESET_DELAY** to a value of **0.5 or higher**.

Windows:

```
set BLINKA_MCP2221_RESET_DELAY=0.5
```

Linux:

```
export BLINKA_MCP2221_RESET_DELAY=0.5
```

This is a value in seconds to wait between resetting the MCP2221 and the attempt to reopen it. The reset is seen by the operating system as a hardware disconnect/reconnect. Different operating systems can need different amounts of time to wait after the reconnect before the attempt to reopen. Setting the above environment variable will override the default reset delay time, allowing it to be increased as needed for different setups.

Using FT232H with other FTDI devices.

Blinka uses the libusbk driver to talk to the FT232H directly. If you have other FTDI devices installed that are using the FTDI VCP drivers, you may run into issues. See here for a possible workaround:

https://forums.adafruit.com/viewtopic.php?f=19&t=166999 (https://adafru.it/doW)

Getting "no backend available" with pyusb on Windows

This is probably only an issue for older versions of Windows. If you run into something like this, see this issue thread:

https://github.com/pyusb/pyusb/issues/120 (https://adafru.it/Uao)

which describes copying the 32bit and 64bit DLLs into specific folders. (example for Win7 (https://adafru.it/Uao))

Getting "no backend available" or other problems with pyusb on Mac

Check out this issue thread:

https://github.com/pyusb/pyusb/issues/355 (https://adafru.it/19fh)

which has **lots** of discussion. It is probably worth reading through it all to determine what applies for your setup. Most solutions seem to rely on setting the **DYLD_LIBRARY_PATH** environment variable.

This issue thread has further information:

https://github.com/orgs/Homebrew/discussions/3424 (https://adafru.it/19fi)

I can't get neopixel, analogio, audioio, rotaryio, displayio or pulseio to work!

Some CircuitPython modules like may not be supported.

- Most SBCs do not have analog inputs so there is no `analogio`
- Few SBCs have `neopixel` support so that is only available on Raspberry Pi (and any others that have low level neopixel protocol writing
- Rotary encoders (`rotaryio`) is handled by interrupts on microcontrollers, and is not supported on SBCs at this time
- Likewise `pulseio` PWM support is not supported on many SBCs, and if it is, it will not support a carrier wave (Infrared transmission)
- For display usage, we suggest using python `Pillow` library or `Pygame`, we do not have `displayio` support

We aim to have, at a minimum, `digitalio` and `busio` (I2C/SPI). This lets you use the vast number of driver libraries

For analog inputs, the MCP3xxx library (https://adafru.it/CPN) will give you `AnalogIn` objects. For PWM outputs, try the PCA9685 (https://adafru.it/tZF). For audio, use pygame or other Python3 libraries to play audio.

Some libraries, like Adafruit_CircuitPython_DHT (https://adafru.it/Beq) will try to bit-bang if pulsein isn't available. Slow linux boards (<700MHz) may not be able to read the pins fast enough), you'll just have to try!

Help, I'm getting the message "error while loading shared libraries: libgpiod.so.2: cannot open shared object file: No such file or directory"

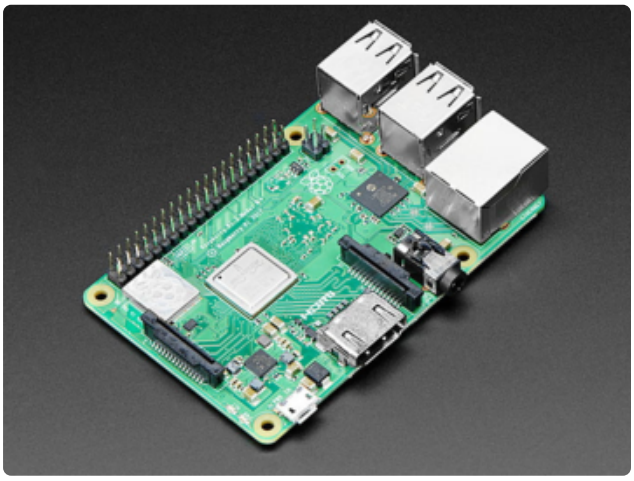
It looks like **libgpiod** may not be installed on your board.

Try running the command: `sudo apt-get install libgpiod2`

= v5.5.0""> When running the libgpiod script, I see the message: configure: error: "libgpiod needs linux headers version >= v5.5.0"

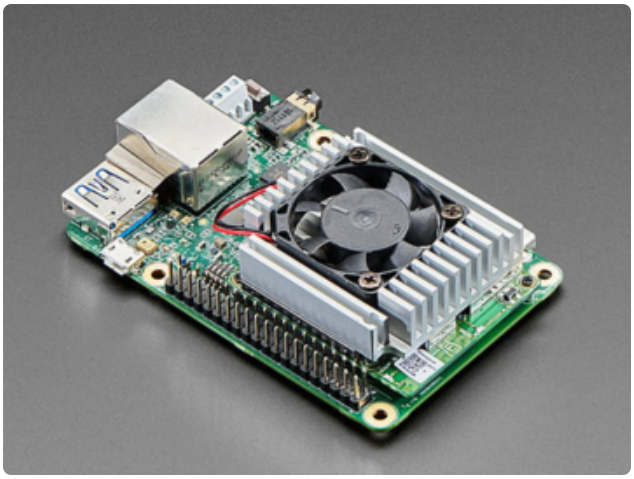
Be sure you have the latest libgpiod.py script and run it with the `-l` or `--legacy` flag:

```
sudo python3 libgpiod.py --legacy
```



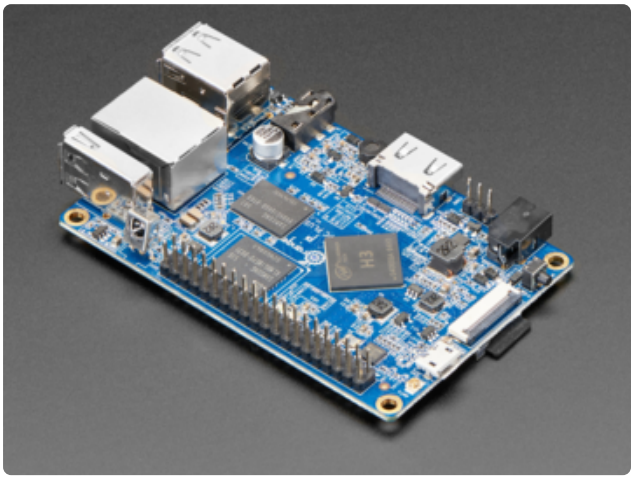
All Raspberry Pi Computers Have:

1 x I2C port with **busio** (but clock stretching is not supported in hardware, so you must set the I2C bus speed to 10KHz to 'fix it')
2 x SPI ports with **busio**
1 x UART port with **serial** - note this is shared with the hardware console
pulseio.pulseIn using gpiod
neopixel support on a few pins
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



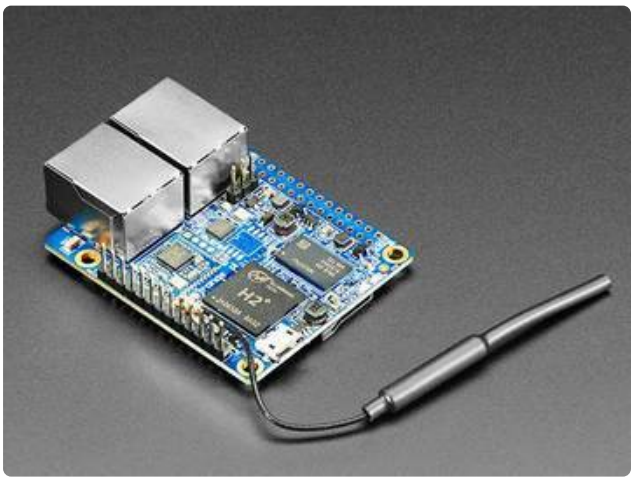
Google Coral TPU Dev Boards Have:

1 x I2C port with **busio**
1 x SPI ports with **busio**
1 x UART port with **serial** - note this is shared with the hardware console
3 x PWMOut support
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)



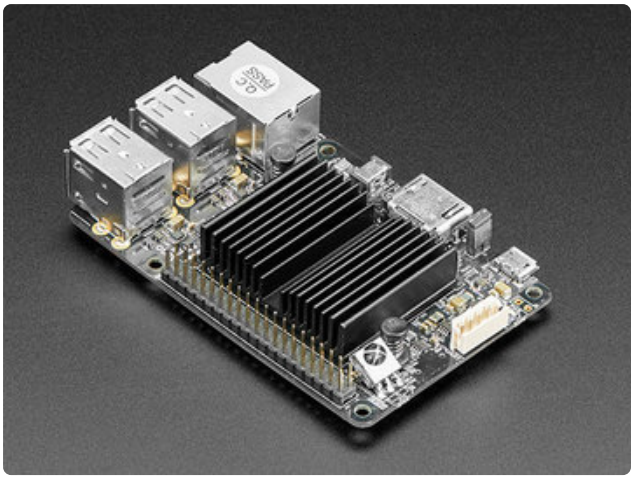
Orange Pi PC Plus Boards Have:

1 x I2C port with **busio**
1 x SPI ports with **busio**
1 x UART port with **serial**
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



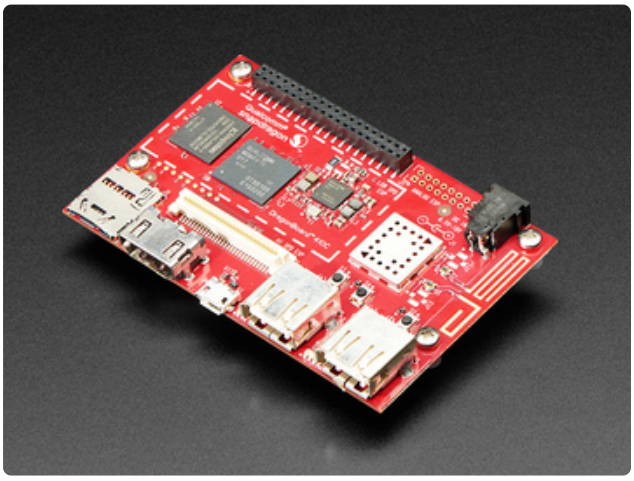
Orange Pi R1 Boards Have:

1 x I2C port with **busio**
1 x SPI port with **busio**
1 x UART port with **serial**
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



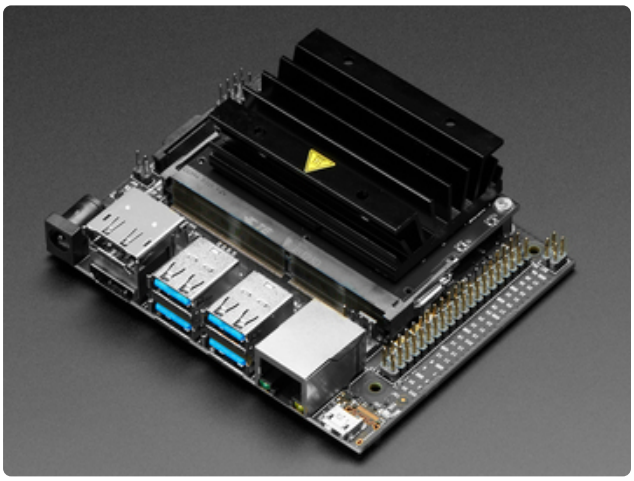
ODROID C2 Boards Have:

1 x I2C port with **busio**
No SPI support
1 x UART port with **serial** - note this is shared with the hardware console
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



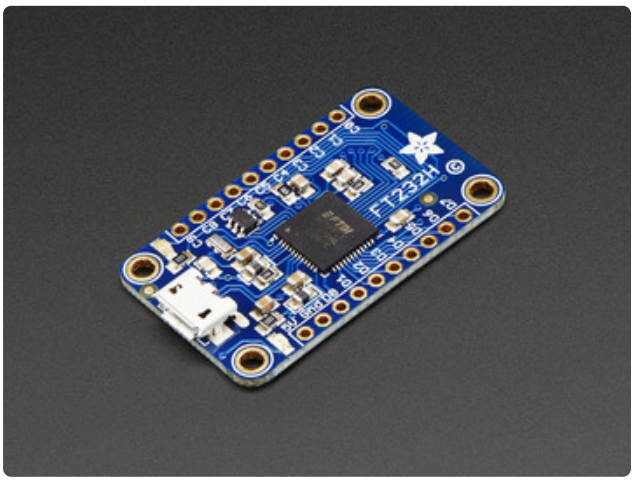
DragonBoard 410c Boards Have:

2 x I2C port with **busio**
1 x SPI port with **busio**
1 x UART port with **serial**
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



NVIDIA Jetson Nano Boards Have:

2 x I2C port with **busio**
2 x SPI ports with **busio**
2 x UART port with **serial** - note one of these is shared with the hardware console
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)

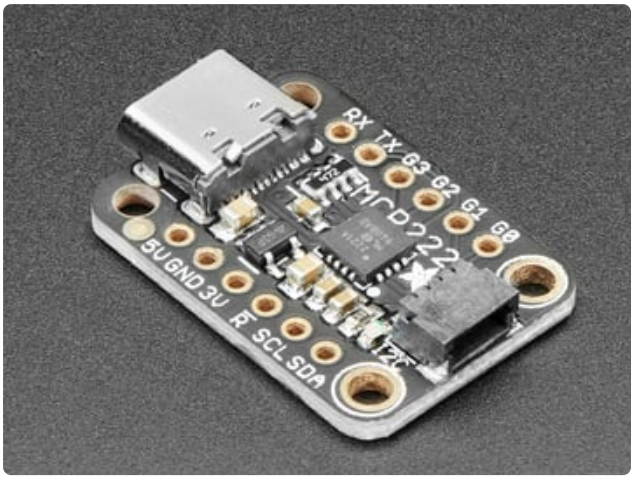


FT232H Breakouts Have:

1x I2C port **OR** SPI port with **busio**
12x GPIO pins with **digitalio**
No UART
No AnalogIn support
No AnalogOut support
No PWM support

If you are using Blinka in FT232H mode (https://adafru.it/FWD), then keep in mind these basic limitations.

SPI and I2C can not be used at the same time since they share the same pins.
GPIO speed is not super fast, so trying to do arbitrary bit bang like things may run into speed issues.
There are no ADCs.
There are no DACs.
UART is not available (its a different FTDI mode)



MCP2221 Breakouts Have:

1x I2C port with **busio**
4x GPIO pins with **digitalio**
3x AnalogIn with **analogio**
1x AnalogOut with **analogio**
1x UART with **pyserial**
No PWM support
No hardware SPI support

If you are using Blinka in MCP2221 mode, then keep in mind these basic limitations.

GPIO speed is not super fast, so trying to do arbitrary bit bang like things may run into speed issues.
UART is available via `pyserial`, the serial COM port shows up as a second USB device during enumeration